Universidade de Aveiro
2014

Departamento de Eletrónica,
Telecomunicações e Informática

Tiago Jorge de Oliveira Magalhães

VitalRemote: Solução *in loco* para Gestão de Sistemas Ciber-Físicos

VitalRemote: *in loco* Solution for Management of Cyber-Physical Systems

Universidade de Aveiro
2014

Departamento de Eletrónica,
Telecomunicações e Informática

Tiago Jorge de Oliveira Magalhães

VitalRemote: Solução *in loco* para Gestão de Sistemas Ciber-Físicos

VitalRemote: *in loco* Solution for Management of Cyber-Physical Systems

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor José Maria Amaral Fernandes, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor João Paulo Silva Cunha, Professor associado c/ agregação do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto.

Dedico este trabalho aos meus pais, namorada e amigos.

o júri / the jury

presidente / president

Prof. Doutor Rui Luís Andrade Aguiar
professor associado com agregação do Instituto de Telecomunicações da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor José Maria Amaral Fernandes
professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Prof. Doutor Pedro Miguel Alves Brandão
professor auxiliar do Departamento de Ciência de Computadores, Faculdade de Ciências da Universidade do Porto

agradecimentos /
acknowledgements

Em primeiro lugar gostaria de agradecer ao meu orientador, Professor Doutor José Maria Fernandes, por todo o apoio, técnico e moral, que me levou a conseguir concluir este trabalho. Pela sua disponibilidade, simpatia e excelente carácter humano. Ao meu co-orientador, Professor Doutor João Paulo Silva, pela oportunidade de realizar este trabalho assim como pela disponibilização de material técnico.

Um obrigado a todos os colegas de laboratório pela sua sempre boa disposição e ajuda. Por todos os concelhos e vivências partilhados que me ajudaram a crescer ao longo deste ano.

Agradeço toda a ajuda dos meus colegas e companheiros ao longo destes anos, principalmente ao Eng. Diogo Regateiro pelos vários momentos, experiências, vivências e conhecimentos partilhados. Por saber sempre como desvanecer a minha ansiedade com as suas palavras e postura calma e com a certeza de que sem ele estes anos seriam muito mais complicados.

Também quero deixar um agradecimento muito especial aos meus pais, Alvarim e Margarida Magalhães, por toda a disciplina e educação transmitidas, por diariamente lidarem comigo, nomeadamente nos dias menos bons, e pelos constantes incentivos. Sem eles toda a minha vida académica não teria sido possível.

A ti, Patrícia, pela enorme paciência e compreensão demonstradas e pelos incentivos e motivação que me deste ao longo destes anos.

Palavras Chave sistemas ciber-físicos, gestão & controlo *in loco*, dispositivos móveis, Android.

Resumo

Dispositivos, como Smartphones ou tablets, estão cada vez mais presentes nas nossas vidas, interligando os mundos cibernético e físico formando Sistemas Ciber-Físicos (SCF). Contudo, a gestão e configuração de tais sistemas tem-se tornado um problema: tanto pelo tamanho reduzido dos sensores e atuadores, fornecendo uma fraca ou nenhuma interface para o utilizador, como pelo seu grande número. Bons exemplos destes SCF são os dispositivos vestíveis e embutidos nos têxteis.

Nesta dissertação propomos o VitalRemote, uma solução que permite configurar, monitorar e gerir SCF. O VitalRemote fornece um simples protocolo de *Handshake* sem fios (e.g. Bluetooth, NFC) para configuração básica in loco e uma interface baseada em REST, usando Wi-Fi, para configurações remotas mais complexas. O VitalRemote estabelece diversas abstrações para os recursos disponíveis nos dispositvos (hardware, sensores, ou aplicações e serviços internos) e permite a configuração baseada em perfis através do mapeamento de instruções em REST para comandos específicos do dispositivo.

A atual implementação do VitalRemote é baseada em Android e foi aplicada num cenário de monitorização de profissionais de emergência usando a aplicação DroidJacket sem acesso direto aos dispositivos de monitorização. Neste cenário a re-engenharia da solução DroidJacket permitiu que esta fosse reconvertida num dispositivo compatível com o VitalRemote. Usando clientes externos do VitalRemote em Android e em iOS, o sistema foi testado com sucesso usando várias combinações de diferentes dispositivos Android e diferentes soluções de conectividade (Wi-Fi e rede *ad-hoc*).

O VitalRemote provou ser uma solução simples para gerir, configurar e controlar SCF remotamente.

Keywords

cyber-physical systems, *in loco* management & control, mobile devices, Android.

Abstract

Devices, such as Smartphones or tablets, are increasingly present in our daily lives, interlinking the cyber and physical worlds in the so-called Cyber-Physical Systems (CPS). However the management and configuration of such systems are becoming a problem: either due to the reduced size of the sensors and actuators, that provide poor or none human user interface, or by the sheer number of them. Good examples are the wearable and embedded devices.

In this dissertation we propose VitalRemote, a solution that allows configuring, monitoring and managing CPS. VitalRemote provides a simple wireless Handshake protocol (e.g. Bluetooth, NFC) for *in loco* basic configuration and a REST based interface for more complex remote configurations, over Wi-Fi. VitalRemote establishes several abstractions for available resources (own device hardware, sensors or internal applications and services) that allow profile-based device configuration like mapping specific REST instructions to custom resources related commands.

The current VitalRemote implementation is based on Android and was applied in a scenario of first responders monitoring using the DroidJacket system where no direct access to the device was possible. In this scenario the reengineering of DroidJacket allowed it to be converted into a VitalRemote compliant device. Using external client in Android and iOS, the new system was tested with success using different Android devices and different connectivity solution (Wi-Fi and ad-hoc network).

VitalRemote proved to be an unobtrusive solution to manage, configure and control CPS remotely.

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

API	Application Programming Interface
BAN	Body Area Network
CPS	Cyber-Physical Systems
DIM	Domain Information Model
ECG	Electrocardiogram
GPS	Global Positioning System
HDMI	High-Definition Multimedia Interface
ICE	Integrated Clinical Environment
IP	Internet Protocol
JSON	JavaScript Object Notation
MAC	Media Access Control
MVC	Model-View-Controller
NFC	Near Field Communication
ObjC	Objective-C
PDA	Personal digital assistant
PPE	Personal Protective Equipment
REST	Representational state transfer
RFID	Radio-Frequency Identification
SOA	Service-Oriented Architecture
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
UUID	Universally Unique Identifier
XML	Extensible Markup Language
WAN	Wide Area Network
WBAN	Wireless Body Area Network
Wi-Fi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network

CHAPTER 1

Introduction

About 20 years ago, Mark Weiser [1] envisioned that pervasive devices and services would become parts of our daily lives. Devices, such as Smartphones, e-readers, GPS-enabled cameras, tablet computers, and other gadgets, are changing the way we interact in our ecosystem by interlinking the cyber and physical worlds [2] leading to the concept of Cyber-Physical Systems (CPS) . CPS are defined by the intersection [3] of the cyber world, composed by computing devices, and the physical world, where they can act as sensors and actuators.

The interactions between the cyber and physical worlds (illustrated in Fig. 1.1) include sensing the real world (through sensors) and/or actuating over it through actuators according with specified parameters. For example, in a system for urban flood management [4], sensors, distributed around a city, measure the present water level, that is relayed to central control room that, according with configured parameters, decides on how to operate the floodgates, on the water channels using actuators.

CPS are present in many application domains, such as healthcare monitoring, aviation safety and monitoring, automotive collision avoidance, or energy management systems. However, their management and configuration is not straightforward namely due to the variety and number of sensors and the absence of a human user interface (e.g. small or in wearable devices and embedded devices). Consequently, in real scenarios, there is a need of a quick and unobtrusive solution to manage and configure CPS [5].

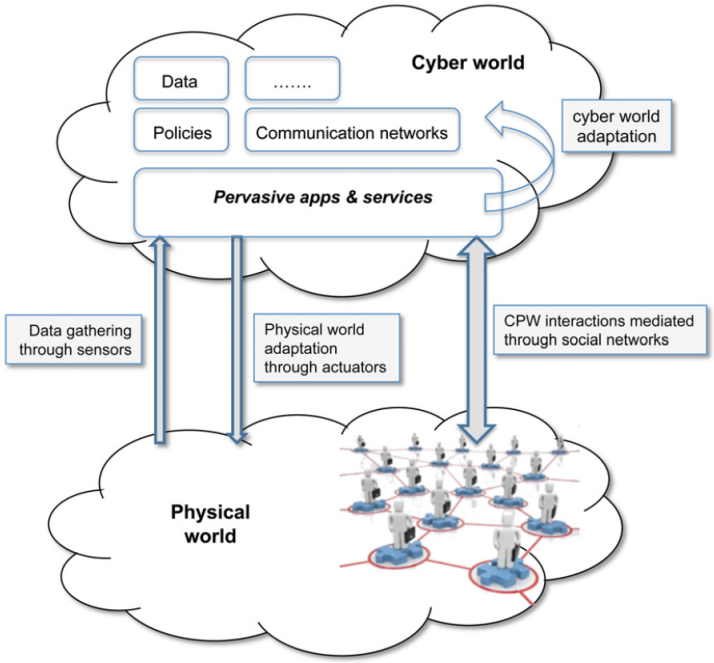


Figure 1.1: Physical and cyber world interactions (from [2]). The sensors collect information about physical reality which are used and processed by devices in cyber world in order to monitor and control the physical world, using actuators.

1.1 OBJECTIVES

Motivated by a specific first responder monitoring CPS called DroidJacket [6], our main objective is to provide a solution that allows wireless managing, configuring and controlling of generic CPS in similar conditions – seen as set/network of sensors. In this scenario, relying on physical cable connection or specific docking stations is not suited for these as they interfere with tasks at hand especially and because CPS devices may not be easily accessible (e.g. incorporated in the gear or under protective garment). A solution for these scenarios should provide:

- A CPS abstraction that allows the system architecture to be flexible to accommodate existing and future sensing solutions, identifying each node and assigning roles.
- Standard communication interfaces for assessing the devices and respective configurations.
- Allow runtime change in the CPS namely replacing existing physical sensors/nodes without compromising the overall configuration and integrity of the system.

In this dissertation we propose VitalRemote. VitalRemote is a system that allows wireless configuration and monitoring of one or more devices. VitalRemote provides two configuration

interfaces: the Handshake interface (Bluetooth, Near Field Communication (NFC)), used in *in loco* basic configurations, such as enable or disable device's resources, and a Representational state transfer (REST) based interface for more complex remote configurations, over Wireless Fidelity (Wi-Fi) . VitalRemote defines a generic device template which allows abstracting resources typically found in CPS resources namely in their own device hardware (e.g. Bluetooth, Global Positioning System (GPS)), paired sensors, as logical data streams providers, or the installed applications and services, as custom CPS resources. Furthermore, VitalRemote was designed to allow onsite change/replacement of paired sensors by equivalent ones i.e. same data stream protocol.

1.2 DISSERTATION STRUCTURE

This dissertation is organized in six chapters.

Chapter 1 presents the motivation and objectives of the present work that lead to VitalRemote.

Chapter 2 describes the related State of Art containing an overview on Cyber-Physical Systems and main application domains illustrated by some existing projects/commercial products. Still in this chapter, is presented the ISO/IEEE 11073 standard which defines a set of sub-standards and a system architecture to guarantee interoperability between medical devices. In the State of Art we also describe DroidJacket, a solution of monitoring of fire fighters, which was chosen as proof of concept scenario for VitalRemote.

Chapter 3 presents the VitalRemote architecture with emphasis on its rationale and main concepts. In this chapter a detailed description on the communication interfaces and protocol is also provided.

Chapter 4 describes the VitalRemote proof of concept implementation in a fire fighting scenario. We describe the process of DroidJacket system refactoring and the implementation options. Also, we present the test scenarios, the used devices and the results.

Chapter 5 discusses and presents the accomplishments of this work as well as future work in order to give continuity to this project.

CHAPTER 2

State of art

With the evolution of the embedded systems, we have assisted to their size reduction and improvement on computing power and, consequently, became increasingly more present in people's life. An embedded system is a part of a complete device or system to perform specific functions, sometimes under real-time computing rules [7].

The merging between embedded systems and communication capabilities made possible the connections between cyber and physical worlds [2] transforming the traditional embedded system into a more adaptive systems often called today as Cyber-Physical Systems.

2.1 CYBER-PHYSICAL SYSTEMS

The concept of a Cyber-Physical Systems is broad, it involves control, communication and computation [7], and, for this reason, it is hard to find a clear definition. Rajkumar, Insup, Lui, *et al.* [8] in a recent paper define as "physical and engineered systems whose operations are monitored, controlled, coordinated, and integrated by a computing and communication core". Several works [3], [7], [9] consider CPS as real-time control systems which are adaptive, networked and/or distributed and possibly with human-in-the-loop, i.e. controlled by humans.

CPS are composed by embedded devices, such as sensors and actuators, which are connected to monitor, sense, and operate physical elements in the real world according with user-defined semantic laws [10], such as time, or pre-defined parameters, managed by cyber world. The interactions between physical and cyber worlds are the most characterized aspects CPS as Edward and Sanjit refer in [11] "CPS is about the intersection, not the union, of the

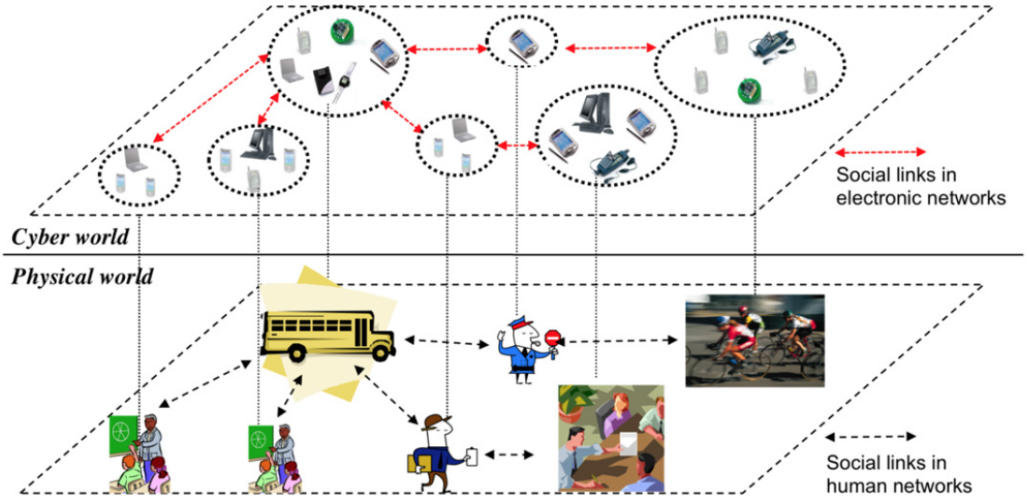


Figure 2.1: CPS integrate the cyber and physical world namely humans and their personal devices and environment (from [2]).

physical and the cyber", in contrast with traditional embedded systems that are "coupled" to computing elements.

Conceptually there is a clear distinction between cyber and the physical worlds in the CPS (Fig. 2.1): physical world englobes humans, the devices and sensors, such as accelerometers or gyroscopes, that sense or act on the real world and exchange information with the cyber world.

However, CPS are not limited to these simplified representations and often involve concerns related with quality attributes like security, resilience and privacy that go beyond a simple technical perspective - Fig. 2.2 presents a conceptual map that offers a good sum-up of the broad number perspectives possible on CPS. For that reason, CPS systems should require design methodologies in order to guarantee, besides validation and verification, that attributes like scalability and ease of complexity management are met [9].

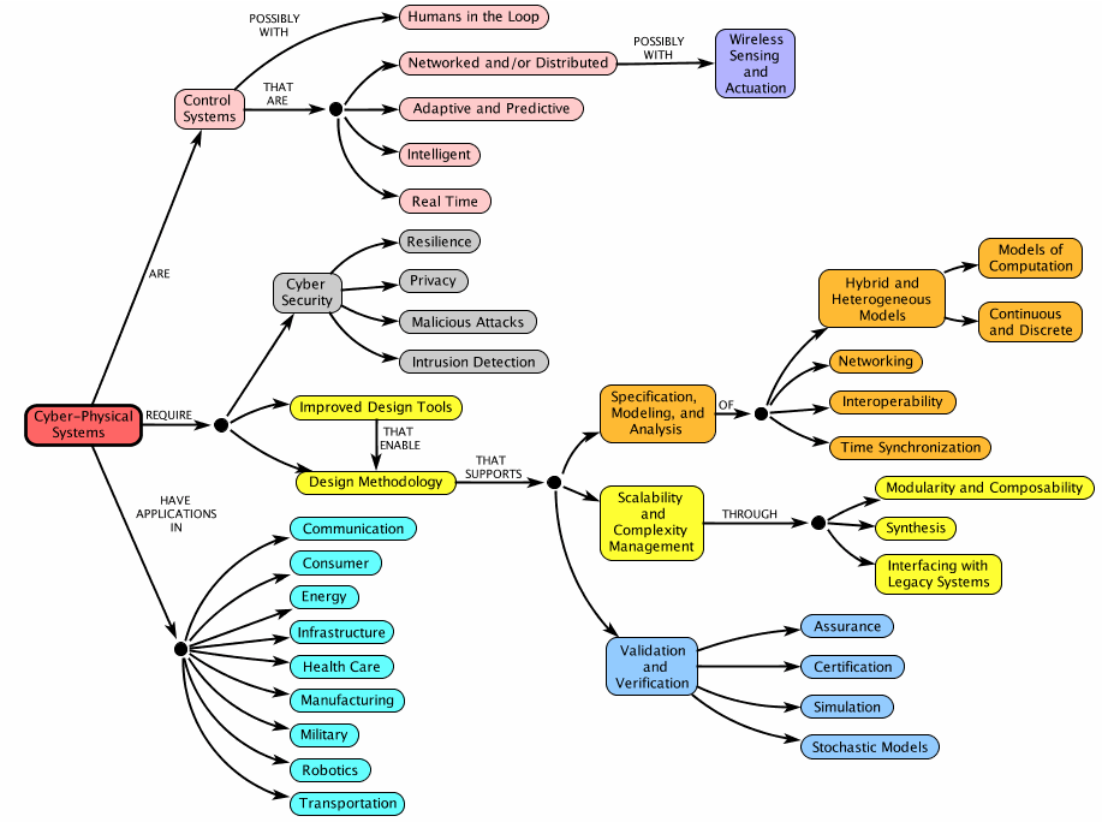


Figure 2.2: Cyber-Physical Systems conceptual map (from [12]).

2.1.1 CPS ARCHITECTURE

As the system involves the real and the cyber world, CPS need to have a support architecture to integrate all the components and entities in order to sense, act, store and process. Different authors present an architecture to CPS. Tan, Goddard, and Perez [10] propose an Event/Information driven architecture using a Publish/Subscribe scheme in which, according to the system goal, subscribe to interesting events/information and publish event/information when necessary. Events are defined by "raw facts" reported by sensor units and "actions" made by actuator units and information is characterized by the abstraction of the physical world made by CPS control units through events processing.

In [13] Wu, Kao, and Tseng suggest an architecture based in three main domains: (1) sensor cooperation, composed by static/mobile sensor and actuator networks, (2) heterogeneous information flow, with storage and applications platforms, and (3) intelligent decision/actuation, containing computing and intelligent decision systems.

A service-based is another type of architecture. Hyun Jung and Soo Dong [14] present a three-based architecture composed by Environmental tier, with physical devices and end-users using the devices and their associated physical environment, Service tier, a typical computing

environment with service in Service-Oriented Architecture (SOA) and Cloud Computing, and Control tier, which receives the gathered data, makes controlling decisions, finds the correct services, and lets the services be invoked on the physical devices. Liang, Nannan, Zhejun, *et al.* [15] have a different vision of service-based architecture. They present an architecture with five tiers: (1) perceive tier, called also sensor tier once it works as data source for the above tiers, (2) data tier, composed by computational and storage devices, (3) service tier, providing the typical functions, (4) execution tier, which receives commands and executes them through actuators, and (5) security assurance, responsible for guaranteeing the system security from unauthorized access or malicious attacks.

According with Sztipanovits, Koutsoukos, Karsai, *et al.* [16] a CPS is composed by three different design layers which are the physical layer, envolving the physical components and their interactions, software layer, comprising the software components, and the network/platform layer, containing the hardware side and including the network architecture and computing platform that interact with the physical components. However, Akanmu, Anumba, and Messner [17] propose a model architecture that provides a good CPS abstraction (Fig. 2.3).

This architecture is composed by multiple layers, each one with its function:

- Sensing layer - where the sensors are. The sensors are responsible for sensing and monitoring some aspects of the real world, e.g. environment temperature. The gathered data is sent to the connected device.
- Device layer – where the devices are. The sensors are connected to a device. A device, such as Personal digital assistants (PDAs), tablets, iPads, or Smartphones, is the mean for the end user to interact with the system and is responsible to allow access to the gathered data from the sensing layer and the entry of user input through an application graphical interface.
- Communication layer – this layer is composed by the Internet and wireless communication networks. The Fig. 2.3 shows examples of possible networks, such as WLAN, WPAN (e.g Bluetooth), WAN, and WiMAX, and each one connects the device, in device layer, to other devices in order to allow sharing of the collected data. The communication layer also allows transfering, through the Internet, these data to be saved in a database present in the contents and application layer.
- Contents and application layer - the contents and application layer contains the database where the gathered data is stored. These data are analyzed and constantly updated to the received one from the communication and actuation layers. This layer also contains

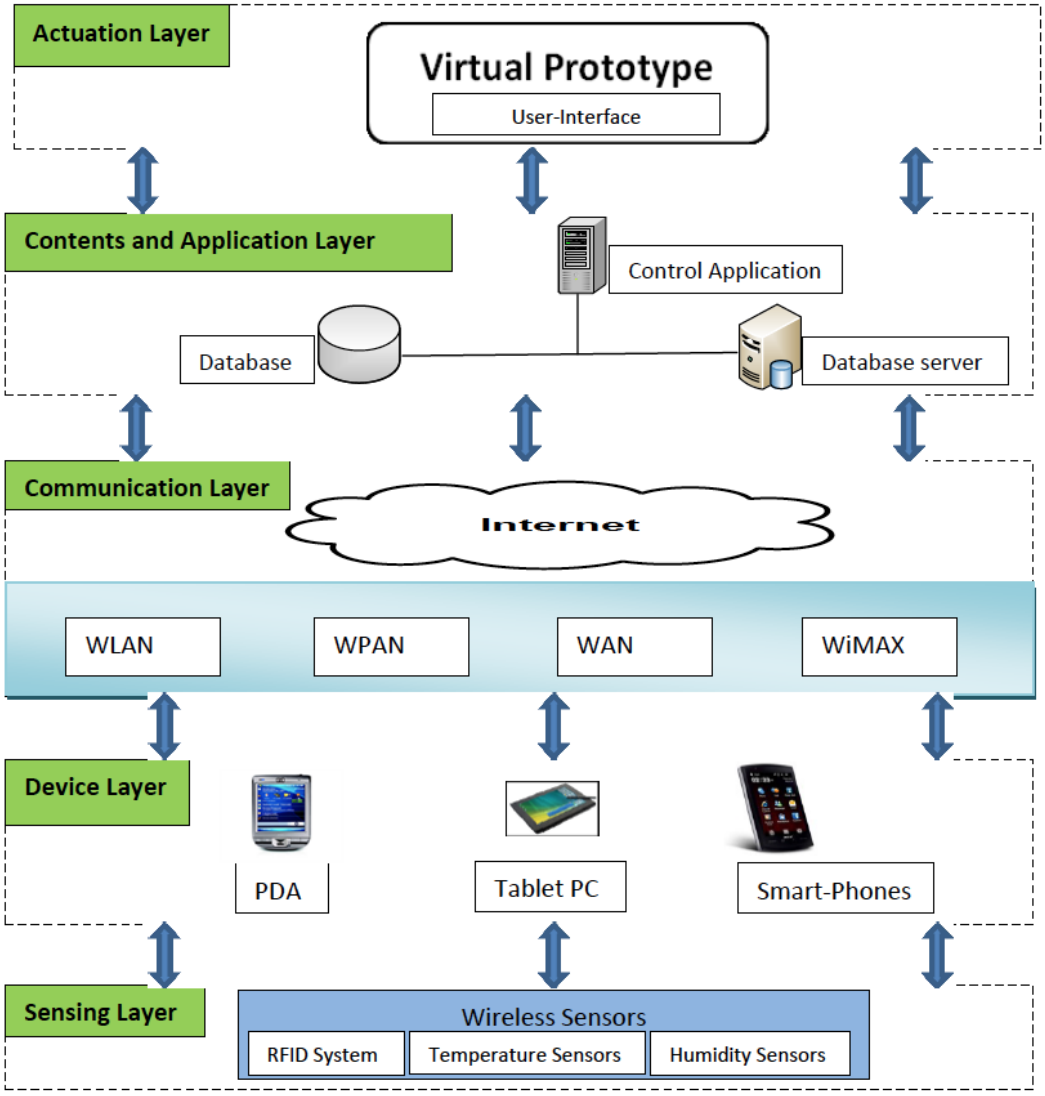


Figure 2.3: Cyber-Physical Systems architecture (as in [17]). CPS are composed by five layers: 1) sensing layer, responsible for monitoring and sensing physical elements of the real world; 2) device layer, whose devices gather data from the sensing layer and allow access to them; 3) communication layer, providing a communication network between devices and capabilities of the 4) content and application layer, which contains a database and control application, and 5) actuation layer, providing an user interface in order to monitor the gathered data and actuate according to eventual anomalies.

the control application which type depends on the context of the system.

- Actuation layer - this layer contains a user interface to permit the user to visualize and monitor the gathered data allowing him to make decisions about the system and actuate according to the anomalies. The decisions are reflected through the various layers once this system is bi-directional.

2.1.2 MAIN APPLICATION DOMAINS

CPS can be found in many domains such as aerospace, healthcare, transportation, and energy management.

In aerospace, CPS research not only has importance on the aircrafts design but also on air traffic management in order to improve aviation safety [7], such as the ActionWebs project [18] at University of California. Baheti and Gill [19] enumerate some key research issues focused on improving the CPS: the tradeoffs between sometimes incompatible qualities (capacity, safety, efficiency, and the interoperability) in systems like integrated flight deck systems; vehicle health monitoring and management; and safety research about aircraft control systems.

Transportation and automotive are another main CPS domains. Current vehicles already have a set of subsystems that are connected in order to control and automate engine control, break system, airbag deployment system, windshield wiper, door locks [7], among others. At same time, vehicles are starting to be able to connect to each other, via Internet, cellular networks, or vehicle-to-vehicle networks, in order to detect traffic congestion and avoid collision [20]. A good example is the GrooveNet [20], [21] tool built at Carnegie Mellon University that implements the algorithms and protocols developed by the project team. Among these algorithms and protocols, the investigators designed a vehicular protocol based on vehicle-to-vehicle communications, GPS, and other automotive sensors in order to enable safer traffic intersection navigation and autonomous vehicle control.

Another major CPS domain is the energy management systems, also called smart power grid. These systems aim to develop a stable advanced power network with real-time control and monitoring of energy consumption [8] in order to reduce energy use [22], and consequently reduce the costs of the users, and improve energy efficiency [7], [9].

For example, Weiss, Mattern, Graml, *et al.* [23] introduce a simple system to monitoring the energy consumption of household devices and they focus on the simplicity of the solution and on the user interaction (Fig. 2.4). They guarantee that comparing with similar or commercial products their product is simpler needing only to install a smart electricity meter. The smart electricity meter is responsible for measuring and logging the energy consumption of the attached devices to the electric circuit of the household. A user can visualize, in real-time, the electric consumption of his house using a web browser or a mobile application developed in Objective-C, that are connected to a gateway, running a Web Server. The communication to the gateway follows a REST approach, i.e. the requests are Uniform Resource Locators (URLs) and the resources are identified by an Uniform Resource Identifier (URI) , and the server

Figure 2.4: User interface measuring the consumption of a device (from [23]).

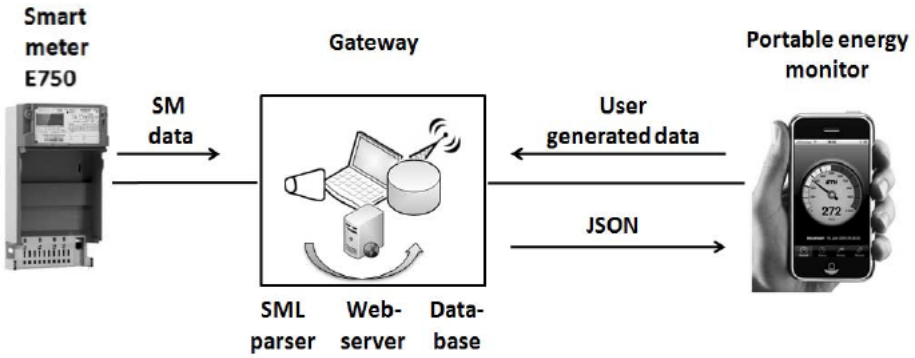


Figure 2.5: Smart meter communicating with the mobile UI (from [23]).

responds with JavaScript Object Notation (JSON) [24] (Fig. 2.5). This system enables a set of functionalities such as obtain the status of the gateway components, restart it, list all smart meters, create resources and get measurements.

2.1.3 CPS in healthcare, and related IEEE standards

We will focus with more detail in the healthcare domain CPS as it somewhat related with the focus of our target scenario – a CPS for monitoring physiological signs from a team of firefighters – the DroidJacket system [6], [25]. The physiological sensors clearly fall within the healthcare area and some features on healthcare CPS may be of value to our present work.

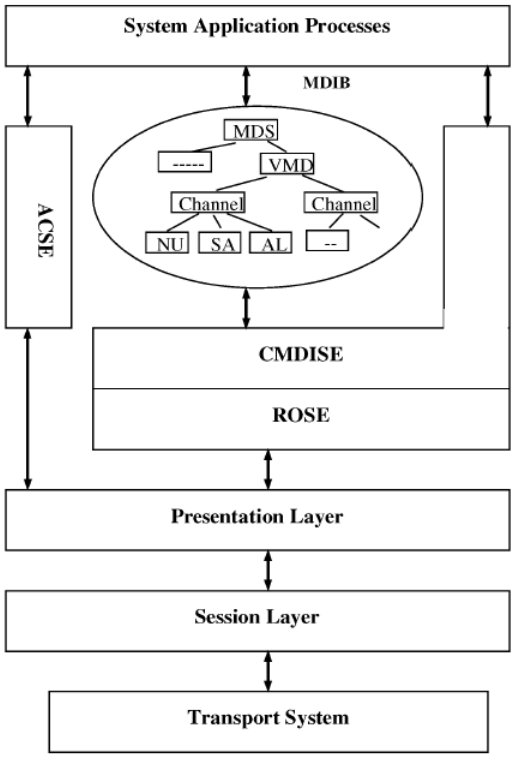
In healthcare, CPSs provide new opportunities and challenges [7]:

- wireless connectivity among medical devices, in order to combat the trend to rely on proprietary communication protocols [3] and create open systems (allow interconnection and interoperation with other systems) [26] of a larger scale and complexity [3].

- reliability and safety, which can be compromised by "malignant spagheti", i.e. a mesh of cables from various devices.
- to improve patient safety and health care efficiency. Lee and Sokolsky [3] suggest the use of Medical Device Plug-and-Play initiative [27] in order to have a safe and flexible interconnectivity of medical devices.
- to assure that devices can communicate, control, and interact under strict timing, determinism, energy and (re)configurability constraints [26] the U.S. Food and Drugs Administration [28] developed an official guideline for medical wireless network development. In paper [3] Lee and Sokolsky reffered the use of open interconnectivity standards, such as Integrated Clinical Environment (ICE) standard, for Medical Cyber-Physical Systems.
- developing new abstraction techniques for creating patient models, which are used to design of closed-loop control and safety analysis of scenarios, such as model drug absortion by the patient body where relationships between drug dose and concentration and patient vital signs, such as heart and respiratory rates, must be taken into consideration, and adaptive patient-specific algorithms and smart alarms [3], once the medical devices are designed to trigger an alarm in conditions of an "average" patient.
- real-time physiological sensing, allowing a safer and more capable [26] monitoring of vital signs and physical activities remotelly [3], [28] using wearable sensors [26] and body sensor networks [3].
- ensure security in Medical CPS networks and privacy of the electronic health records [3], [28].
- verification, validation and certification. These aspects aim to make more robust [28] and, consequently, trustworthy CPS [26].

ISO/IEEE 11073 HEALTH INFORMATICS - MEDICAL / HEALTH DEVICE COMMUNICATION STANDARDS

In the healthcare domain there have been efforts to overcome traditional problems of interoperability between devices. In the medical and health devices area there are many manufacturers and each one producing its own communication protocol and interfaces meaning that devices from different proprietaries can hardly communicate, i.e. the devices are not interoperable. Among these efforts, IEEE proposed the ISO/IEEE 11073 (a.k.a X73) standard, a family of standards. For the IEEE, the interoperability is “the ability of two or more



ACSE:	Association Control Service Element
AL:	Alarm (Channel)
CMDISE:	Common Medical Device Information Service Element
MDIB:	Medical Data Information Base
MDS:	Medical Device System
NU:	Numeric (Channel)
ROSE:	Remote Operation Service Element
SA:	Sample Array (Channel)
VMD:	Virtual Medical Device

Figure 2.6: Conceptual model for an X73-compliant medical device (from [30]).

systems or components to exchange information and to use the information that has been exchanged" [29]. In order to reach the global interoperability between medical devices and external computer systems the devices must be capable to interpret correctly the exchanged messages, in terms of nomenclature, data types, message syntax, and encoding rules. As a result, ISO/IEEE 11073 comprises the seven-layer ISO/OSI model (application, presentation, session, transport, network, data link, and physical) [30], as illustrated in Fig. 2.6, which aim to solve the described problems.

The X73 conceptual model covers all the layers and, in this model, the System Application Processes uses the Association Control Services and Medical Device Information Services to connect to other devices and to access managed objects in the Medical Data Information Base, which can be stored locally or remotely [30]. The standard follows an object-oriented system management paradigm whose Domain Information Model (DIM) [31], defined in ISO 1173-10201, which specifies objects, attributes, attribute groups, event reports, and communication services, used to communicate device data and to configure medical devices and functionalities. Every item exchanged between systems is identified by a numeric code which is defined by the standardized nomenclature (ISO 11073-10101) [29].

Yao and Warren [30] give an example of applying the ISO/IEEE 11073 Standards to

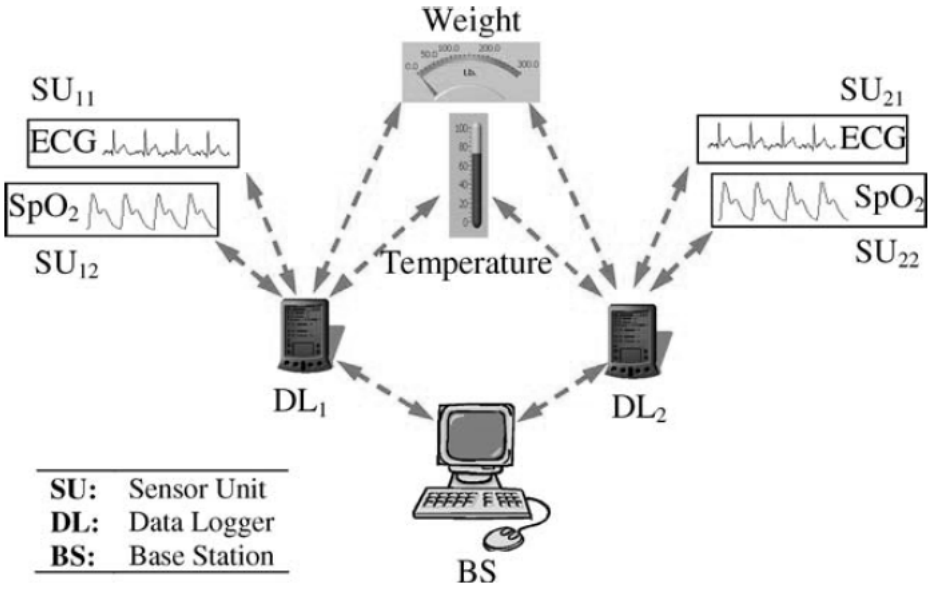


Figure 2.7: Architecture of a health monitoring system constructed with X73 components (from [30]).

Wearable Home Health Monitoring Systems (illustrated in Fig. 2.7).

The system constructed with X73 standards has several components: (1) Sensor Units (SUs), that can be wearable sensors which are used by only one person and nearby sensors, such as weight scale or ambient temperature sensor, (2) Data Loggers (DLs), a mobile device that stores the user data before relaying it to a monitoring station, and (3) Base Stations (BSs), that process, display, store, and forward the received data to remote clients over the Internet. These three device categories must follow the presented conceptual model (Fig. 2.6) to guarantee the interoperability between devices from different manufacturers. From the X73 standards point of view, an agent (SUs) uses a Device Communication Controller (DCC) and a manager (BSs) uses a Bedside Communication Controller (BCC) to interact with their transport system. A DL works as agent and manager, i.e. uses both DCC and BCC, to communicate with its SUs and BS, respectively.

2.2 SOME CPS CASE STUDIES

CPS are constantly in developing and new solutions are appearing with different objectives, such as people health monitoring, construction industry security, energy consumption controlling, or driving safety. In this section we present some interesting CPS examples that have some intersections with our current work namely in the overall architecture design, in

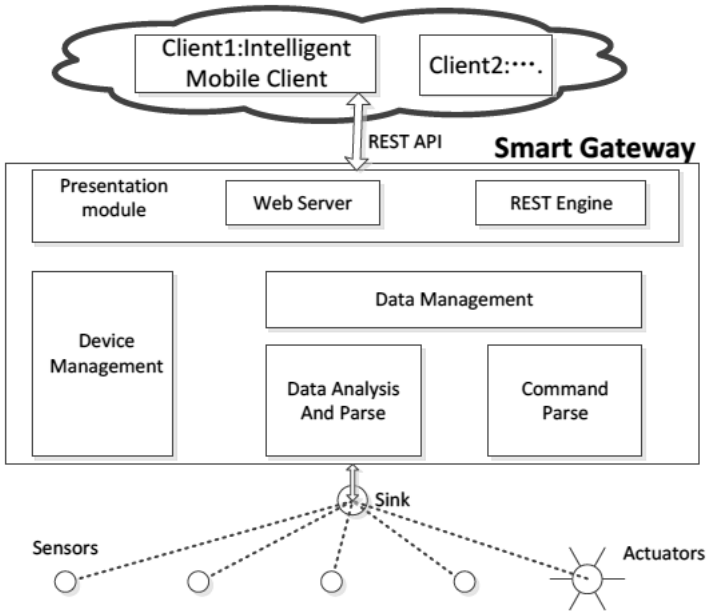


Figure 2.8: Restful Smart Gateway Architecture (from [32]).

the configuration and monitoring solution and/or purpose.

2.2.1 RESTFUL SMART GATEWAY

The Smart Gateway [32] provides a vision of a centralized architecture for a management and control system of a group of sensors and actuators. This architecture makes use of a Restful Application Programming Interface (API) for clients' interactions and Extensible Markup Language (XML) and JSON messages' codification as server response. The article also provides the tests' results about delay time of the referred messages' codifications allowing the readers to have a better choice.

Smart Gateway has an architecture based on a REST-style architecture and is composed by several modules (Fig. 2.8). The presentation module is the access point to clients interacting with any available devices and their respective services. This module provides XML and JSON types of message for resource presentations. The Smart Gateway also contains the device management module that maintains a list with the active devices by broadcasting periodically a message.

If the Smart Gateway device management does not receive a response it deletes the item in the list and destroys the generated URL. An URL is generated for each device that can successfully register in the Smart Gateway using a simple protocol based on exchanging a "HELLO" and "ACK"s messages. The data management is composed by the data and

Figure 2.9: User interface of the Smart Gateway client for monitoring surrounding temperature (from [32]).

command parse modules. The Command Parse is responsible for parsing the clicked URLs received by the presentation module and send a command to the sink. The data management also stores the sensors' data received by the sink, through IEEE 802.15.4 [33], i.e. Low-Rate Wireless Personal Area Networks (LR-WPANs).

Any Web browser or the client (Fig. 2.9) are able to access the physical devices and services by clicking URL links through a Restful API.

The Smart Gateway allows its clients to know the active resources, access each one individually, retrieve the history and change the state of the physical devices. Smart Gateway was evaluated in several experiments focused on the time delay from a request to the back response of using XML vs JSON in message codification. They concluded that for small messages both types have a similar delay time but if the data amount is large the JSON messages have a better performance.

Figure 2.10: Protective elements with the prototype embedded (from [34]).

2.2.2 PERSONAL PROTECTIVE EQUIPMENT MONITORING SYSTEM

The Personal Protective Equipment (PPE) monitoring system proposed in [34] uses a wired Body Area Network (BAN) for Radio-Frequency Identification (RFID) tag reading in order to control the correct use of PPE. Each BAN contains a device which transmits, through a mesh network, the readings to the coordinator node which it is connected to a computer, via USB. Besides the storage of the readings, the coordinator database also stores the configurations of the all devices to enable an easier system administration.

To guarantee the security, the workers, in the construction industry, must use and wear certain elements, such as helmets, gloves, boots, goggles, harness, among others [34]. However, due to the workers not using the PPE or using it inappropriately a technician has to inspect the workers to avoid these situations, which proved to be an ineffective solution because they are periodic inspections and not real-time monitoring.

The system is based on a BAN of sensors that are able to detect the presence of a PPE corresponding to a part of the body. Each PPE has an associated RFID tag which is read by one of the many RFID readers presented in the workers' clothing (Fig. 2.10).

The readers transmit the gathered data to a device via a wired connection (Fig. 2.11). This device contains a radio module that is responsible for relaying the information to the coordinator through the closer router of the mesh network (Fig. 2.12).

The coordinator is connected through an Universal Serial Bus (USB) port to the computer that has running the central server software, which processes and stores the received data in a database. Besides the gathered data, the coordinator database also stores the configuration of

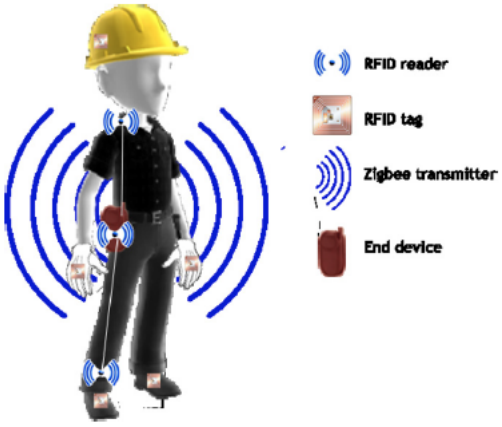


Figure 2.11: Architecture of the system installed on the workers clothings. The connections between the RFID readers and the end device are wired (from [34]).

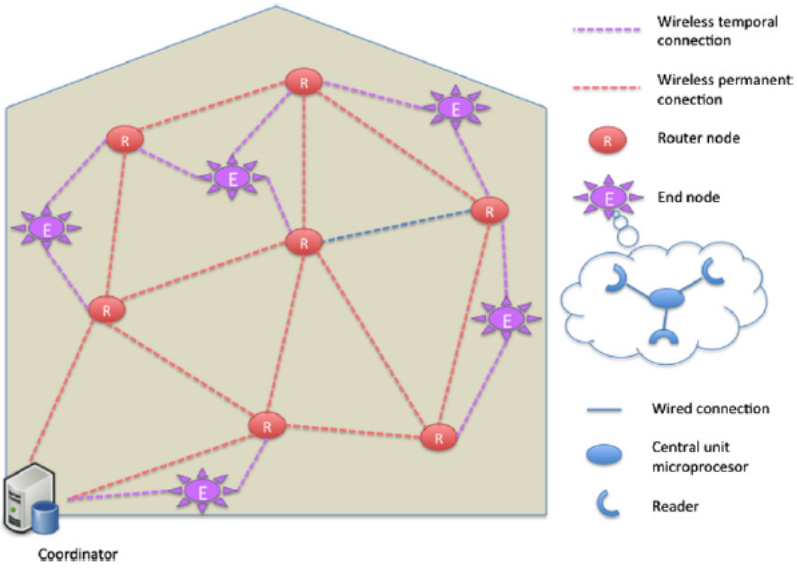


Figure 2.12: Network architecture of the PPE monitoring system (from [34]). A mesh network of router nodes which the end nodes (devices connected to RFID readers) connect to. One of the nodes corresponds to Coordinator, responsible for storing all gathered data and devices' configurations.

all nodes centrally, which allows an easier system administration. The configuration consists of the specification of the monitoring resolution, i.e. how many times the PPE Detection phase is executed per time unit, number of PPEs and their respective IDs. The end node device requests the configuration when it enters in the network for the very first time or when the reset button is pressed, and stores the received configuration for future uses.

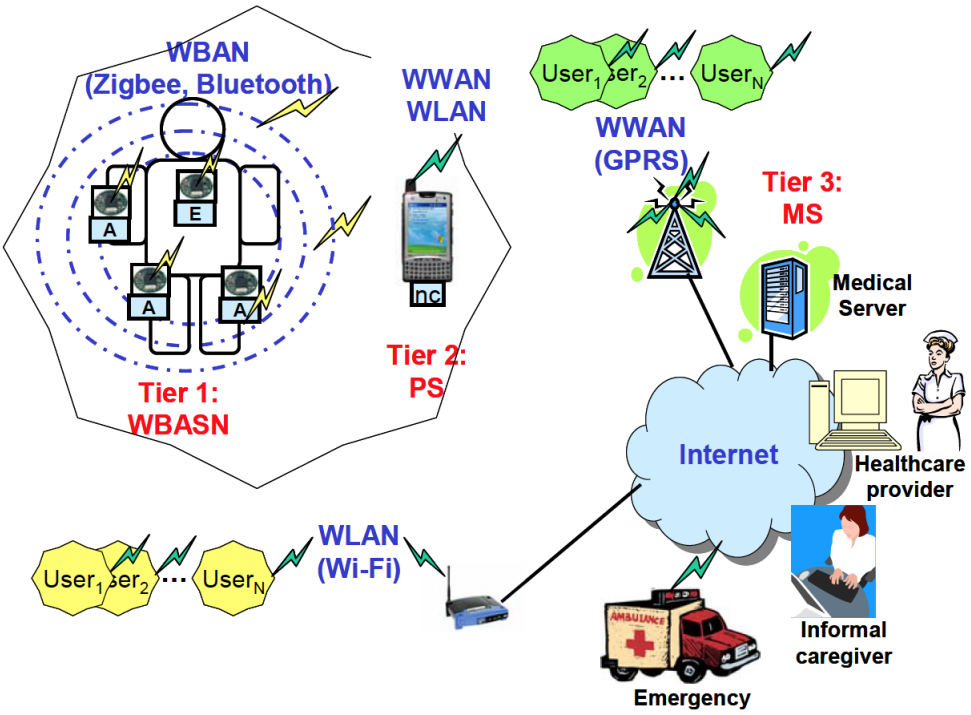


Figure 2.13: Health Monitoring System Network Architecture (from [5]). A user wears several sensors that are connected, via wireless technology, to Personal Server (PS) which transmits the collected information to Medical Server (MS) through WLAN (Wi-Fi) or WWAN (GPRS).

2.2.3 WIRELESS BODY AREA NETWORK FOR HEALTH MONITORING

The Health Monitoring System proposed in [5] consists of many Wireless Body Area Networks (WBANs), which gather vital data of the individual who is using it. In this system each person carries a device (PDA, cell phone, or home personal computer) which is connected to a Network Coordinator, via USB. The Network Coordinator implements a ZigBee and Bluetooth interfaces to communicate and configure the sensors. Besides the Internet networks this system also uses mobile telephone networks (GPRS, 3G) to relay the gathered data to a medical server.

The authors described the software architecture of the Health Monitoring System for WBAN and how it can be integrated into a broader telemedical system. In the proposed architecture (Fig. 2.13) a patient wears a collection of sensors nodes which gather vital signs and relay them to the Personal Server through the connected Network Coordinator (Fig. 2.14a), via USB, that implements ZigBee and Bluetooth interfaces.

Besides to provide a graphical or audio interface to the user, the Personal Server, such as PDA, cell phone, or home personal computer, makes use of Internet or mobile telephone networks (e.g. GPRS, 3G) technologies to transfer the gathered health data to the medical

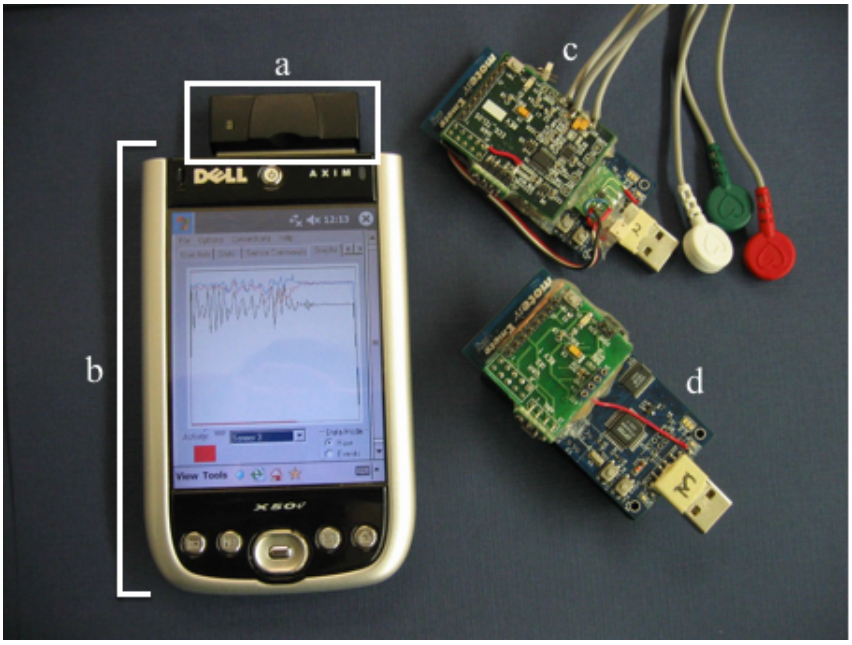


Figure 2.14: Health Monitoring System prototype WBAN (adapted from [5]). In this prototype is present the (b) Personal Server connected to (a) Network Coordinator, (c) ECG sensor with electrodes, and a (d) motion sensor .

server. The Personal Server is also used to configure the WBAN, such as sensor node registration (type and number), initialization (e.g. specify sampling frequency and mode of operation), customization (e.g. run user-specific calibration or user-specific signal processing procedure upload), and setup of a secure communication (key exchange).

In this system it is the responsibility of the medical server to perform authentication tasks and data processing in order to recognize health anomalies and contact and alert the respective caregivers. The medical server also can send instructions to the patient from, for instance, the patient's physician who can inspect the data via Internet.

2.3 VITALRESPONDER: DROIDJACKET APPLICATION

VitalResponder is a project that aims to monitor vital signals of first responders and, for this, its team developed an Android application, the DroidJacket (Fig. 2.15b), that uses VitalJacket [1] [35] (Fig. 2.15a), a wearable technology capable of gather vital signs, such as ECG, to monitor first responders in critical emergency scenarios.

In VitalResponder [6] (Fig. 2.16), the first responders have to wear VitalJacket and run DroidJacket, which is able to connect to multiple VitalJackets, i.e. sets of sensors,

[1] `http://www.vitaljacket.com/`

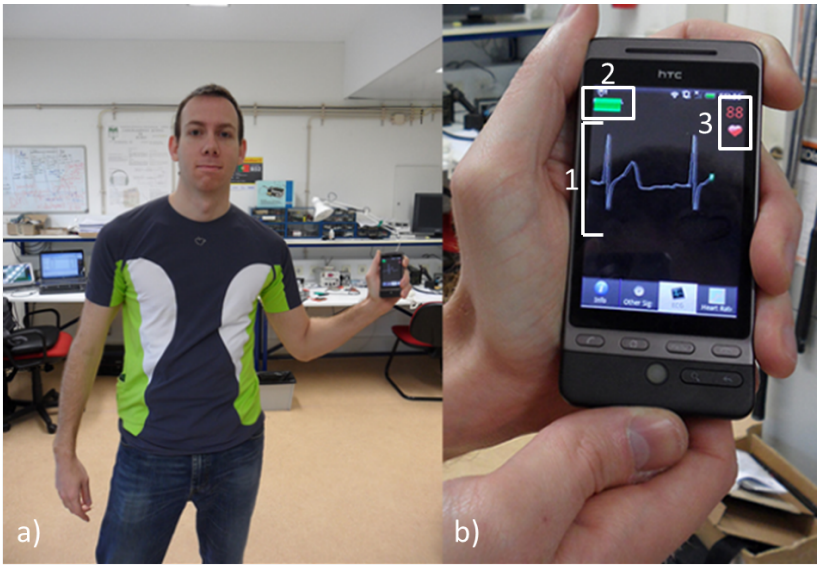


Figure 2.15: VitalJacket (a) and DroidJacket application (b) displaying the (1) ECG graph, (2) ECG sensor battery and (3) heart rate (from [35]). VitalJacket contains sensors, such as ECG electrodes that measure vital signs, which DroidJacket receives and display them.

simultaneously, via Bluetooth, and visualize the gathered data on the device. These data also can be sent to an upper layer central point (Central Operator/Operators), which contains storage capabilities. The stored data will be available to perform offline biomedical signal processing algorithms.

DroidJacket takes in account multitasking and background services to connect to the various sensors and to process simultaneously [25], triggering alarms and notifications in abnormal situations, such as arrhythmias. Since 2010, DroidJacket is evolving and new sensors and functionalities were added, such as the VitalAir [36], [37].

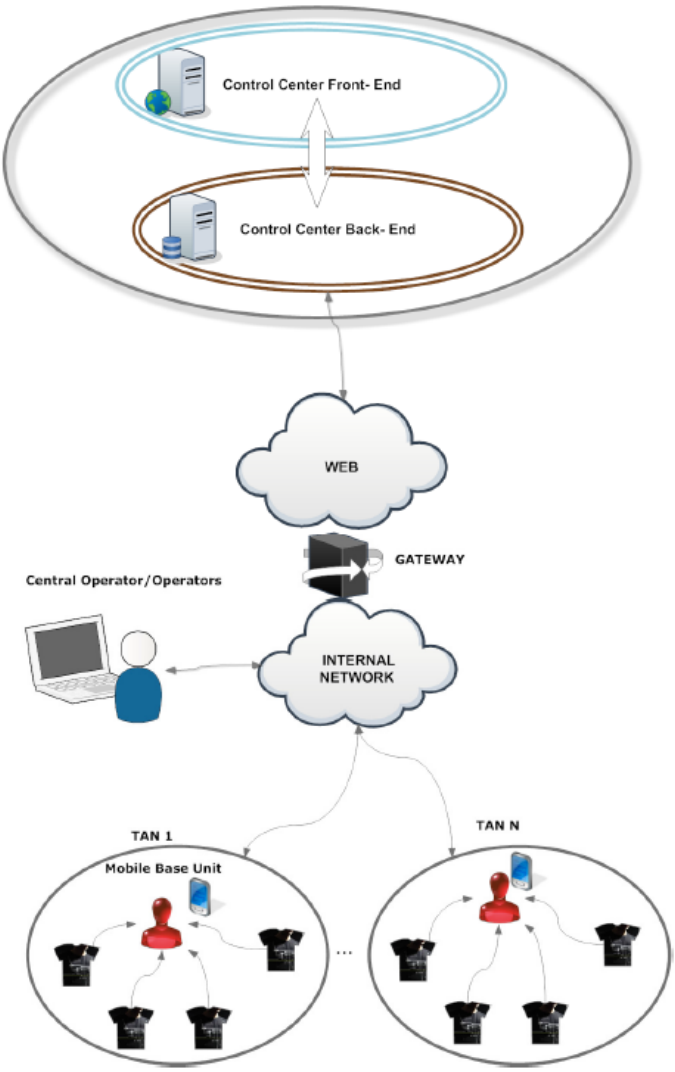


Figure 2.16: Approach to VitalResponder network (from [6]). In each Team Area Network (TAN) the Mobile Base Unit (running DroidJacket) receives the gathered data of VitalJackets and sends it to an upper layer central point (Central Operator) with storage capabilities.

CHAPTER 3

VitalRemote system

In this chapter VitalRemote is presented. VitalRemote is a system that allows wireless configuration, monitoring and device management (via Bluetooth, NFC, or Wi-Fi). This allows a CPS management avoiding the need of a direct contact with the CPS devices, either cabled or using a user interface. This chapter focus on VitalRemote's architecture and rationale.

3.1 VITALREMOTE ARCHITECTURE

In VitalRemote system architecture the VitalRemote device concept, personalized by a Smartphone in the Fig. 3.1, is a central point. The VitalRemote device exposes to the outside three types of resources, its own hardware resources, such as Bluetooth or GPS, logical resources, like temperature sensor paired via Bluetooth, and installed applications that can be launched, and it is responsible for handling directly with all of them.

In order to configure, control, and manage them, the device is always waiting for new connections from client applications, represented by the tablet in the figure. A client application can use two different protocols to configure the device: the Handshake protocol, used for basic configurations, or the REST interface protocol, for advanced ones. Moreover, the device also allows to the user to map custom REST messages to application specific commands. This allows to extend the system functionalities and it is what allows the VitalRemote system to be totally adaptive for any system or problem.

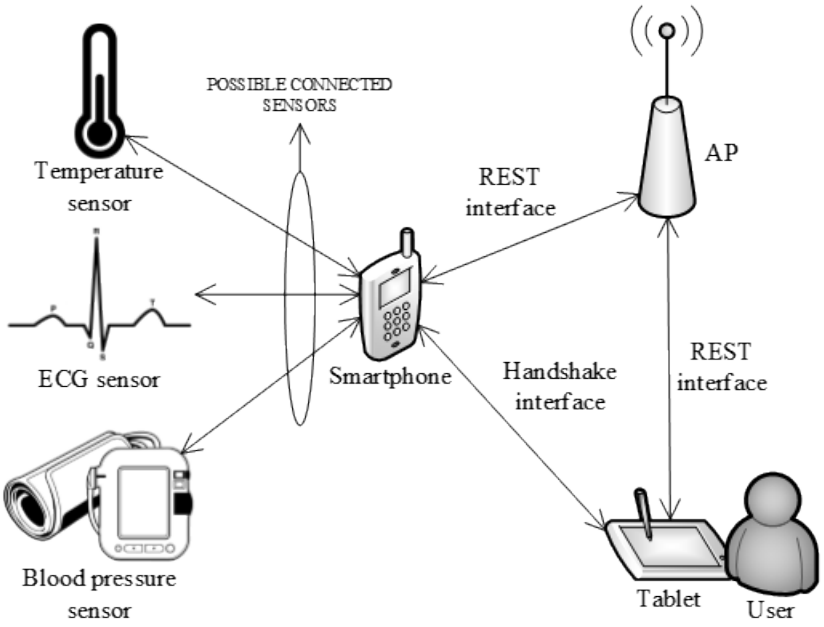


Figure 3.1: Conceptual VitalRemote system overview. VitalRemote device (Smartphone) is the central point of the architecture. It allows that, through REST (Wi-Fi) and Handshake (Bluetooth or NFC) interfaces, a VitalRemote Client (Tablet+user) configures the device's resources and obtain data streams from connected sensors.

3.2 VITALREMOTE DEVICE CONCEPT

VitalRemote defines a generic device concept allowing resources abstraction, namely own device hardware (e.g. Bluetooth or GPS), paired sensors (e.g. via Bluetooth) as logical data streams providers or installed applications and services. Thus VitalRemote can be applied to any type of device that provides Bluetooth and Wi-Fi.

The VitalRemote device is a crucial point in the VitalRemote architecture. It is responsible for receiving configurations and commands and ensuring that they are deployed making it a complex component in the system. To make this possible a VitalRemote device must provide, at least, Bluetooth or NFC capabilities to allow the initial wireless configuration Handshake and support Wi-Fi connectivity for more advanced configurations through the VitalRemote service using a REST based interface.

The device is composed by three big groups of resources (Fig. 3.2):

- the device own hardware resources, such as Bluetooth, GPS, NFC, or Wi-Fi,
- logical sensors, like temperature sensor (paired externally),
- installed applications.

As the device's own hardware resources are specific to the equipment, the developed VitalRemote service must be able to handle with them. Normally, the operating systems such

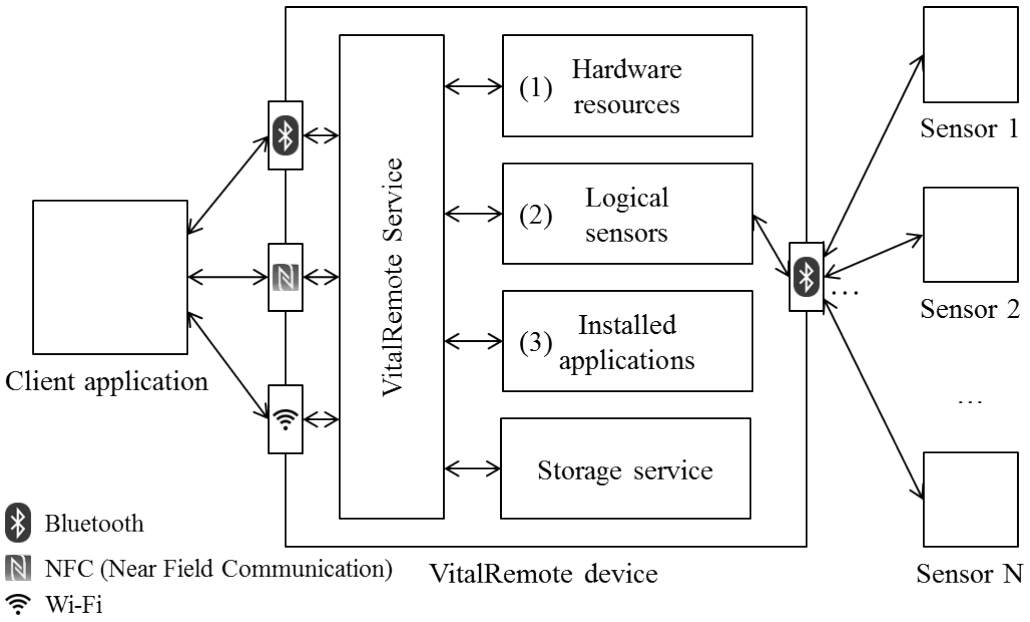


Figure 3.2: A VitalRemote logical device. Besides the service layer to interact with the exterior, a VitalRemote device exposes 3 types of resources: (1) device own hardware resources, (2) logical sensors paired with the device and (3) installed applications that provide custom data streams. A storage service ensures that configurations are persisted in the device.

as Android and iOS provide an abstraction layer with defined APIs which can be used in order to perform basic operations like turn on or off the resource and connect it to a specific device, if it is the case. According to the device's available resources the set of default REST URIs should change.

For VitalRemote, the logical sensors are physical devices with which the device is connected, for example, via Bluetooth. These sensors measure something of the real world, e.g. environment temperature or human Electrocardiogram (ECG) , and provide a continuously stream of the gathered information. It is the responsibility of the VitalRemote service to provide a quick and remote configuration of the connections but not to use and process the received data.

The installed applications can be seen as logical resources once they can process or monitor data and, depending on the application and its goal, they can extend widely the functionalities of the system. These applications are accessed by the REST interface in order to start them remotely and configure their specific parameters. For this, the applications must provide a public interface to execute more complex tasks which must be specified and mapped to the VitalRemote REST interface – in this context the VitalRemote service only acts as a gateway between external clients and the applications running on the device. To store the mapping

between the REST messages and the applications' specific commands the VitalRemote service uses the storage service that the device also must have.

3.3 HANDSHAKE WITH VITALREMOTE

As previously presented there are two different protocols to configure a VitalRemote device. In this section it is presented the Handshake a simple protocol for wireless basic configurations designed for proximity scenarios using Bluetooth or NFC technologies to communicate. It is assumed that both devices have been previously paired avoiding additional stages during the Handshake interactions.

Any external application that needs to interact with a VitalRemote device should follow the wireless Handshake protocol to configure or to initiate contact with the VitalRemote device.

During the Handshake six messages are exchanged (Fig. 3.3): 1) an optional commands mapping file, containing the new client REST URIs to be stored (see section 3.4), 2) a configuration file, containing various configurations to VitalRemote service deploy, 3) the Internet Protocol (IP) address of the data relay monitoring server, i.e. an optional server to debug and validate what it is sent to the monitoring server, and 4) the IP address and

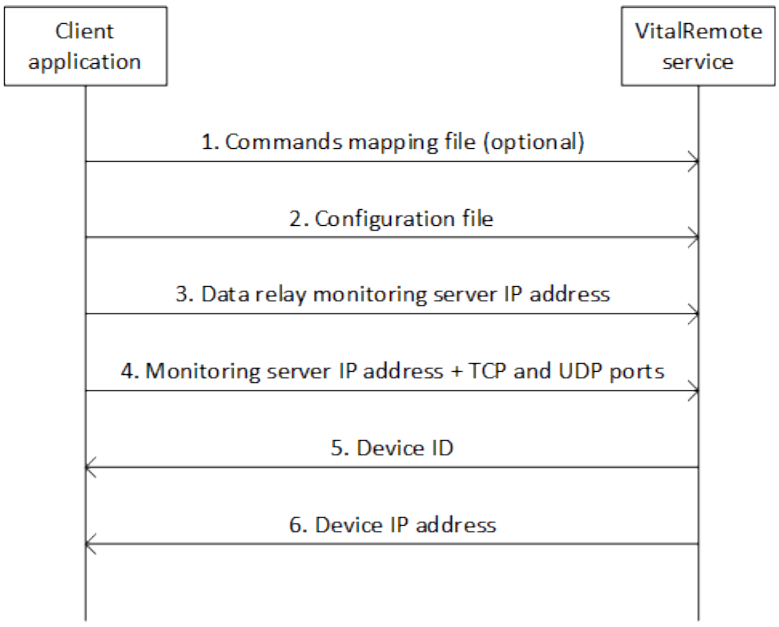


Figure 3.3: Handshake protocol. During the Handshake the client application contacts with VitalRemote passing basic configuration (interactions 1 to 4). VitalRemote returns device specific reference and IP address (interactions 5 and 6)

the respectively Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) listening ports of the monitoring server. Then, the VitalRemote service responds with the 5) device identification, an unique value within the system, e.g. mobile serial number, and 6) its IP address. The retrieved values from the VitalRemote device can be used, for example, to initiate a TCP connection to the device in order to use the REST interface.

After the end message exchange, Handshake connection is terminated and the VitalRemote device becomes again available for new connections and, consequently, for new configurations.

3.3.1 VITALREMOTE CONFIGURATION FILE

A configuration file exchanged during the Handshake can be seen as a script command where the commands correspond to the deployment of each individual configuration setting.

A typical configuration file, as illustrated in code bellow, is composed by two attributes: 1) type, indicating the message type in Handshake phase, and 2) value, containing a set of configurations to be deployed. In this example the hardware resources Bluetooth and Wi-Fi will be turned on, the DroidJacket application will be launched and the ECG data will be relayed via UDP - value 1 on "CONNECTION_MODE".

```
{
    "type": 0,
    "value": [{ "status": true, "type": "BLUETOOTH" },
              { "status": true, "type": "WIFI" },
              { "status": true, "type": "DROIDJACKET" },
              { "status": true, "type": "ECG" },
              { "status": 1, "type": "CONNECTION_MODE" }]
}
```

The device will assume the resources settings that it has available and ignore others that are not available or explicitly referenced. For example, if the configuration above was sent to a system with GPS, GPS status will be maintained, as there is no explicit reference to GPS.

3.4 THE REST INTERFACE

The REST interface allows the VitalRemote clients to configure a VitalRemote device. This interface uses exclusively the Wi-Fi technology and, for this reason, the device must

Num.	Request	Response
1	GET/DEVICE_ID	{ "type":"DEVIDE_ID", "id":"id_value" }
2	GET/DEVICE/HARDWARE_RESOURCES	{ "type":"DEVICE/HARDWARE_RESOURCES", "resources":["BLUETOOTH", "WIFI"] }
3	GET/DEVICE/BLUETOOTH	{ "type":"DEVICE/BLUETOOTH", "status":"[ON|OFF]" }
4	SET/DEVICE/BLUETOOTH/[ON|OFF]	-
5	GET/DEVICE/WIFI	{ "type":"DEVICE/WIFI", "status":"[ON|OFF]" }
6	SET/DEVICE/WIFI/[ON|OFF]	-
7	SHUTDOWN	{ "type":"SHUTDOWN_ACK", "value":"[NACK|ACK]" }

Table 3.1: VitalRemote REST interface requests and sample responses messages.

support Wi-Fi connectivity. The REST interface provides a set of default requests (Table 3.1) that can be used and each one belongs to one of three types: 1) GET to retrieve information from device resources, 2) SET to configure the device resources and 3) SHUTDOWN to finish the connection. A difference to the Handshake protocol is that this interface requires a TCP connection which has to be started and ended by user interaction and while the connection is not finished no other connection is accepted. If a device had two or more connections simultaneously it could receive, at the same time, two or more incompatible messages, like turn on and off the same resource, which, in user terms, would be confusing.

Assuming the VitalRemote device contains Bluetooth and Wi-Fi hardware resources Table 3.1 presents the set of available REST messages. For request 2 the device responds with hardware resources it has. In function of these results, the client application can use messages, such as requests 3 and 5, to obtain the current status of the resource. If the device has more own hardware resources they will be returned and the respective REST commands will be available. Analogously, there are REST commands to turn on or off the hardware resources,

Number	Field	Value type	Description
1	column	String	CP's column to be updated or read.
2	command	String	REST command complying with the described template. Ends without '/'.
3	uri	String	Valid CP's uri in which will be executed the command.
4	value	String	Value to insert in the specified column field.
5	last_segment_value	Boolean (false by default)	Indicate if the value to store is from field value (false) or from last segment of the REST message (true).

Table 3.2: Attributes used for the mapping between REST messages to logical commands.

like requests 4 and 6.

Besides the exposed commands, the system offers to the user the opportunity to create and map REST messages to application specific commands on VitalRemote device. The REST messages must comply with the following template:

<app_package_name>/[APP|GET|SET]/<custom_segments>

The app_package_name guarantees that the commands are unique among different applications that could have the same commands. The following segments differentiate the type of message. APP to start the application with the package name specified in field value, GET to retrieve the value containing in the indicated column and SET to update the column value to a new one. The REST messages have a corresponding application specific command. In our solution's implementation, the commands correspond to an action on a Content Provider, if it is used on an Android OS based device. The mapping file follows the JSON format and rules were specified in order to create the mapping. The Table 3.2 shows the attributes used by the mapping process.

The structure of the mapping file is composed by the message type in the Handshake phase and an array containing the mapping between the REST messages and the logical commands, as shown in the following example:

```
{
  "commands":[...],
  "type":4
}
```

For the user to execute remotely an application, he must use an `APP` REST message type and it is required the indication of the application's package name in the field `value` of the mapping file, as the following example.

```
{
  "command": "pt.ua.ieeta.sias.vr.gui/APP/ON",
  "value": "pt.ua.ieeta.sias.vr.gui"
}
```

In reading scenarios of some value in order, for example, to know if any functionality is active the REST message must to be of the `GET` type and the user must provide the correct Content Provider's `URI` and the respective `column`. The example below reads the value of the column `status` in order to obtain if the service responsible to connect to ECG sensor is executing.

```
{
  "column": "status",
  "command": "pt.ua.ieeta.sias.vr.gui/GET/ECG_SENSOR",
  "uri": "content://pt.ua.ieeta.sias.vr.contentprovider.servicescp/service/ecg"
}
```

In case of `GET` type messages, the VitalRemote service executes the logical command and returns the read value in a JSON file whose `type` attribute contains the REST request. The following example corresponds of a possible response for the previous `GET` REST message.

```
{
   "type":"pt.ua.ieeta.sias.vr.gui/GET/ECG_SENSOR",
   "value":"1"
}
```

Also, it is possible to update a value of a specific column of the Content Provider. For this, the REST command must to be of the `SET` type and the fields `column` and the `uri` must be filled. The value can be static, i.e. indicated in the mapping file in the field `value`, or dynamic, which will be extracted from the last segment of the REST message. In this last type, the user must set the field `last_segment_value` with the value `true`, whose field is

`false` by default.

```
{
  "column": "address",
  "command": "pt.ua.ieeta.sias.vr.gui/SET/ECG_SENSOR_MAC",
  "last_segment_value": true,
  "uri": "content://pt.ua.ieeta.sias.vr.contentprovider.servicescp/service/ecg"
}
```

The example above expresses a case which intends to replace the physical ECG sensor by assigning a new Media Access Control (MAC) address of a different physical sensor to the logical VitalRemote ECG resource. However, this address is only known in runtime. For this reason the address is specified in the last segment of the REST message, e.g. `pt.ua.ieeta.sias.vr.gui/SET/ECG_SENSOR_MAC/00:23:FE:00:06:92`.

The following JSON code shows a portion of the mapping file used in our implementation.

```
{"commands":[
  {"command": "pt.ua.ieeta.sias.vr.gui/APP/ON",
  "value": "pt.ua.ieeta.sias.vr.gui" },
  {"column": "status",
  "command": "pt.ua.ieeta.sias.vr.gui/SET/ECG_SENSOR/ON",
  "uri": "content://pt.ua.ieeta.sias.vr.contentprovider.servicescp/service/ecg",
  "value": "1" },
  {"column": "status",
  "command": "pt.ua.ieeta.sias.vr.gui/SET/ECG_SENSOR/OFF",
  "uri": "content://pt.ua.ieeta.sias.vr.contentprovider.servicescp/service/ecg",
  "value": "0" },
  {"column": "address",
  "command": "pt.ua.ieeta.sias.vr.gui/SET/ECG_SENSOR_MAC",
  "last_segment_value": true,
  "uri": "content://pt.ua.ieeta.sias.vr.contentprovider.servicescp/service/ecg" },
  {"column": "status",
  "command": "pt.ua.ieeta.sias.vr.gui/GET/ECG_SENSOR",
  "uri": "content://pt.ua.ieeta.sias.vr.contentprovider.servicescp/service/ecg" },
],
"type":4
}
```

This example shows five new REST commands that will be stored and mapped to logical commands. The first one corresponds to the execution of the application with the package name `pt.ua.ieeta.sias.vr.gui`, i.e. DroidJacket application. The following two commands serve to enable (value '1') and disable (value '0'), respectively, the Bluetooth connection between DroidJacket and VitalJacket's ECG sensor, controlling the ECG data stream. The fourth command is used when it is intended to switch the ECG sensor. Using this command it is possible to specify in runtime a new Bluetooth MAC address and DroidJacket will, immediately, connect to the new sensor, if the Bluetooth connection is enabled. The last command returns the status of the ECG sensor service, i.e. if DroidJacket is connected to ECG sensor or not.

3.5 VITALREMOTE INTERNAL WORKFLOW

In the previous sections, we exposed and explained VitalRemote architecture, the generic device concept, as well as the management and configuration interfaces. In this section we present the internal workflow of VitalRemote, i.e. how the many components interact between them in order to guarantee the accomplishment of all objectives.

VitalRemote is responsible for listening to clients' connections and deploy all configurations. As described in the previous sections, VitalRemote has two interfaces (Fig. 3.4) to access and configure the device: (3) the REST and (5) the Handshake interfaces. The clients can use the Handshake interface via NFC or Bluetooth while the REST interface is only accessible via Wi-Fi. VitalRemote also contains a (4) Commands Manager which is responsible to manage all the client application specific commands. It uses the (2) Commands Parser to interpret a mapping file, from the Handshake phase, and return a logical command instance in order to be stored in the (1) Storage Service. The Commands Manager also uses the Storage Service to obtain the logical commands when a user intends to use them via REST interface. Each interface is responsible for executing the commands and interact with the (8) applications' Public Interfaces, (6) the Logic Sensors Controller and the (7) Hardware Resources Controller. Both logical sensors and hardware resources are device-specific and the controllers permit to create an abstraction layer allowing to configure, turn on and off, and even add, remove, or replace resources transparently for the rest of system. As the applications already provide a public interface is not necessary to have a component to control the interactions with them.

In the example presented in Fig. 3.5 we describe the Handshake protocol, using Unified

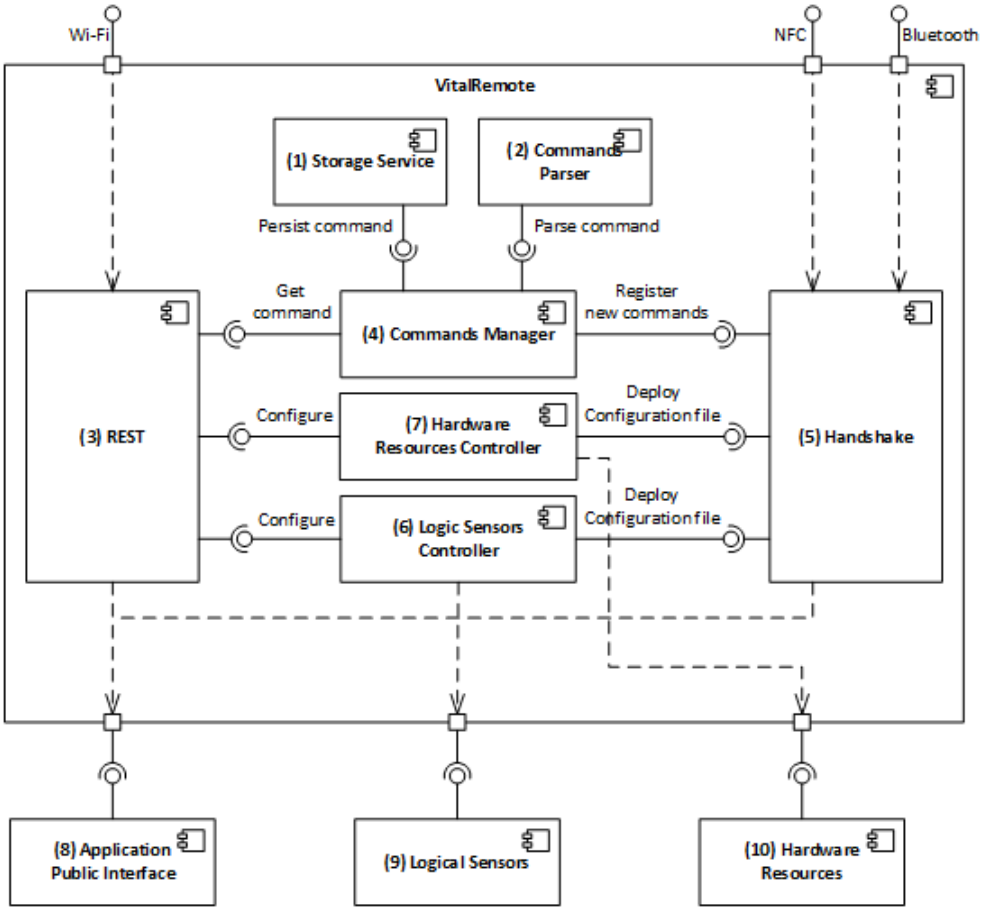


Figure 3.4: Components diagram of VitalRemote. The (5) Handshake and (3) REST interfaces use the (4) Commands Manager to persist and obtain commands stored in (1) Storage Service. The new commands are parsed by (2) Commands Parser before being stored. The (6) Logic Sensors and (7) Hardware Resources controllers create an abstraction layer over the existing paired sensors and hardware resources. The (8) Applications Public Interfaces allows VitalRemote to configure them.

Modeling Language (UML) representation [38]. For sake of readability some interactions are simplified (e.g. between controllers and Storage Service). An external client initiates a Bluetooth connection to a VitalRemote device triggering the Handshake protocol. When the client sends all defined messages the Handshake processes the messages and split them in blocks which are processed in parallel. In case of registering new application specific commands, the Handshake sends the respective block to Commands Manager which parses and stores all messages. At the end of the processing, the Handshake sends the VitalRemote device identification and IP address and finishes the connection.

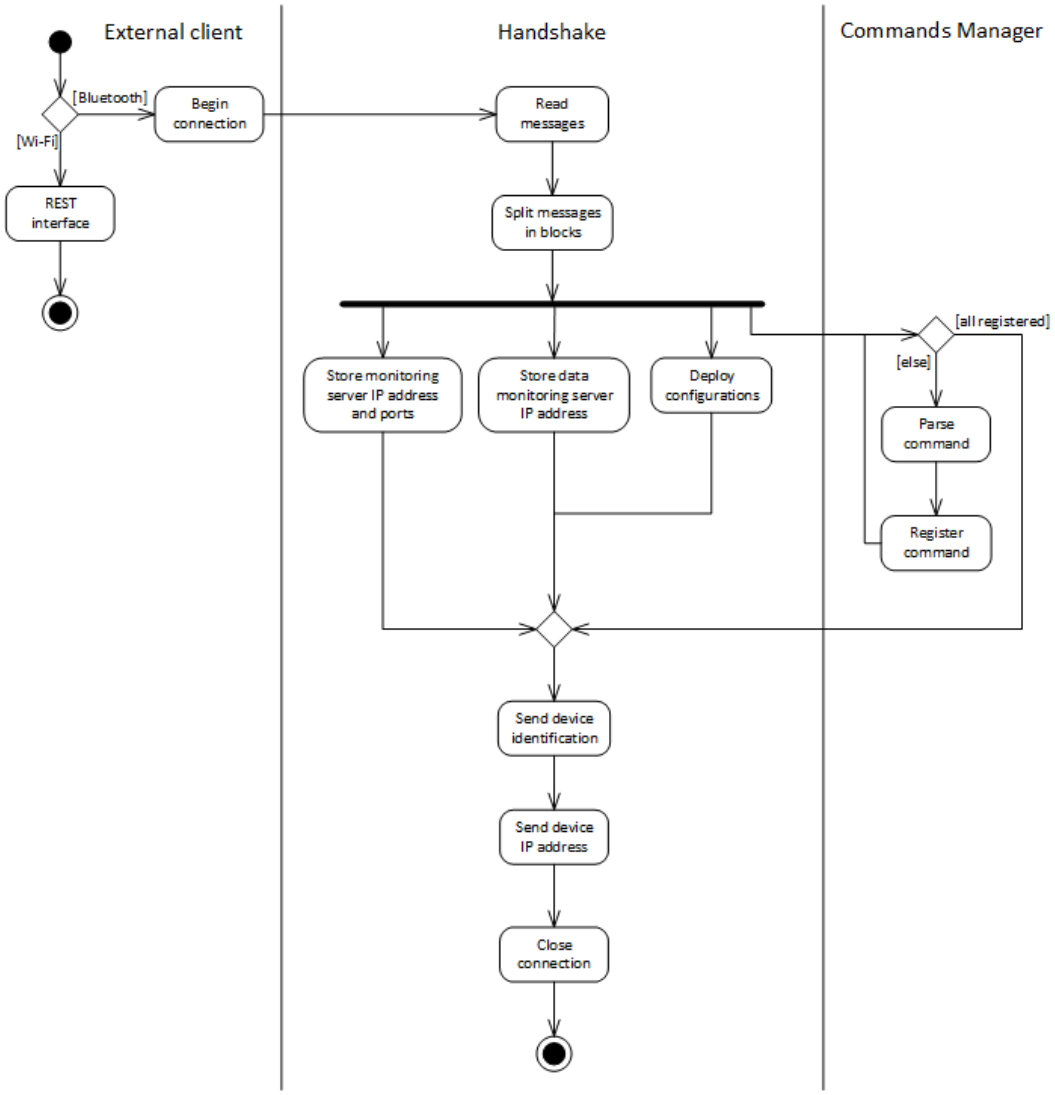


Figure 3.5: Activity diagram of the Handshake protocol. The storages and deploy actions are simplified showing the commands' register process in detail. An external client connects to a VitalRemote device initiating the Handshake protocol. It sends the defined messages which are processed. In case of registering new application specific REST messages the Commands Manager parses and stores them. At the end the Handshake sends VitalRemote device identification and IP address and closes the connection.

CHAPTER 4

VitalRemote implementation

VitalRemote objective was to provide a wireless in loco configuration solution for a Cyber-Physical System. The chosen CPS was the DroidJacket [6], [25], a team monitoring for firefighters during forest fires. The actual VitalRemote implementation was focused on providing a solution to configure DroidJacket when the Smartphones and/or Android PCs devices were kept within the firefighters' protective clothing during the operations.

DroidJacket, besides providing a real scenario, it also allows a proof of concept of the VitalRemote features focused on wireless in loco configuration.

4.1 VitalRemote module

VitalRemote module was implemented as an Android Service. Android services [39] can execute long-running operations in background not needing a graphical user interface. VitalRemote also relies in intents and BroadcastReceivers to trigger actions on its service namely in its boot stage – this is triggered by Android OS own `BOOT_COMPLETED` intent. An application can register to receive broadcasted events, such as battery low event or power connected event. These events are handled by application's BroadcastReceivers, which are only valid during the execution of the call onReceive. When the device is turned on and the Android system is completely booted, it sends `BOOT_COMPLETED` intent and the implemented BroadcastReceiver launches automatically VitalRemote. Moreover, the service

is also listening for Android external changes in the Wi-Fi (`WIFI_STATE_CHANGED` intent) and Bluetooth, related `STATE_CHANGED` intent. These can enable or disable the REST interface and Handshake protocol listening sockets.

VitalRemote also provides a REST interface implemented as a network server node using a ServerSocket instance which, by default, is bound to 11111 port and waits for TCP VitalRemote client connections. When a connection is accepted and the server receives a message it first verifies, using the Commands Manager, if the received message is a custom REST message. If true, the command is executed, otherwise the REST message is parsed and processed configuring the device hardware resources, which in our case were Bluetooth, GPS and Wi-Fi, using the Android APIs. In order to guarantee a larger modularity of the software we developed an entity (Hardware Resources Controller) which is responsible to use Android APIs. Thus, other entities can also configure the hardware resources, without placing repeated lines into its code, making it easier for future software maintenance.

The Handshake component was implemented as Bluetooth server which registers itself with a Universally Unique Identifier (UUID) (a UUID example could be something like 1648fb0a-3096-4efb-bbd2-c26c4b39a174) in Android Bluetooth Service Discovery Protocol (SDP). When a Bluetooth connection is accepted the Handshake protocol is triggered as explained at the end of the section 3.5 (Fig. 3.5). All messages exchanged in Handshake phase follow the JSON codification. To register new application specific REST commands, the Handshake uses the Commands Manager for parsing messages and storing them into an internal database (Storage Service implemented using SQLite). The Handshake component also interacts with VitalRemote compliant applications' Content Providers that allow accessing them as logic resource namely to start, stop and configure them – in the present work DroidJacket's Content Provider is a good example (section 4.2). The DroidJacket application already manages the connections with its clients and, because of that, our work was focused on starting and stopping the data streams and configure network parameters, such as network transport protocol (TCP or UDP), IP addresses and respective ports.

4.2 THE DROIDJACKET REENGINEERING

DroidJacket was conceived as a standalone mobile application and some reengineering was needed to incorporate a VitalRemote compliant configuration interface as a Content Provider. An Android Content Provider is one or more tables containing rows and columns. For

DroidJacket two tables were developed: one for network connections parameters configuration and other, given the service-based design, for control and configurations of DroidJacket's services. Each service is responsible for connecting to its respective sensor and to use the communication module to stream the collected data.

This Content Provider exposes:

- Network transport protocol. For example, in DroidJacket it is possible to choose if the data will be transmitted using a TCP or UDP connection.
- Monitoring server IP address and TCP and UDP ports in order to be possible to change the destination of the streams' messages.
- Data relay monitoring server IP address and stream status. DroidJacket can send the same packets, which are being sent to monitoring server, to other server in order to be possible to verify if DroidJacket is sending and assess and validate the collected values quality. To enable and disable this transmission the stream status value should be changed.
- Service name, to be easier for developers to identify which service corresponds to a table row.
- Service status, to allow enabling and disabling any service, by changing this value.
- Sensor's Bluetooth MAC address, to which the service is or will be connected.

The change of any value provokes a reaction on connection module or service level, e.g. updating the column status, of the table Service, to value "1" corresponds to starting the service and to value "0" to finish it. To switch the Bluetooth MAC address of a sensor to another one it is only necessary to update the column address. This update implies the end of the actual connection and the beginning of a new one to the new address. Analogously, the Network Settings table has the same behavior, i.e. a change on the network transport protocol, for instance, provokes the finishing of the actual connection and beginning of a new one using the updated protocol.

To support UDP messages we had to extend the network connection module in order to also use `DatagramSocket` and `DatagramPacket` classes. Since the VitalRemote client application can monitor many devices simultaneously and they can use UDP connections, which are not connection-oriented, DroidJacket adds the device ID, whose value was transmitted previously in Handshake protocol, into its messages in order to the client application differentiate the VitalRemote device which the message belongs to.

4.3 VITALREMOTE CLIENT

The DroidJacket can be configured to use TCP or UDP transport protocol to provide the various streams and, for this reason, the VitalRemote client contains a TCP and UDP server sockets that are listening for new connections and datagrams, respectively. As proof of concept we developed a VitalRemote client – the RemoteConfigurationTool – as an Android application to be used as a mobile external DroidJacket configuration tool that is described in the next section.

4.3.1 REMOTE CONFIGURATION TOOL

The RemoteConfigurationTool application is able to handle multiple devices configuration, being able to perform multiple Handshakes and multicast the same configuration file to many devices. RemoteConfigurationTool was conceived following the Model-View-Controller (MVC) software architectural pattern, i.e. the application code is divided into three interconnected parts: model, containing the data structures and logic of the application, views, used to visualize the application data, and controllers, to manipulate the model state and to change the view's presentation of the model. Moreover, the application uses the publish-subscribe design pattern to solve the over-processing problem of the received data. The application can be connected to many devices simultaneously and receive different continuous streams from each device but, at a given moment, only one stream, or a subset of them, is visualized. For this reason, we developed a system of publish-subscribe pattern where the visualization fragments register themselves to receive a determined stream of a specific device, using the device ID received in the Handshake phase and which are presented in each message of DroidJacket's streams.

In Fig. 4.1 the application is connected, via Wi-Fi, to a device. On the right side appears the actual state of the VitalRemote device and it is possible to change it by clicking on the buttons. On the left side, we can see some information about the device and about the external sensors it has, such as if they are connected, relaying and, when possible, battery level. On the other hand, the Fig. 4.2 shows the view for ECG signal, heart rate and R-R interval values monitoring. All visualized values were provided by DroidJacket application.

As a proof of concept we also developed a thin configuration tool based on iOS, Fig. 4.3, to turn on/off device hardware resources using directly the Wi-Fi based on TCP connection to the device - without Bluetooth Handshake.

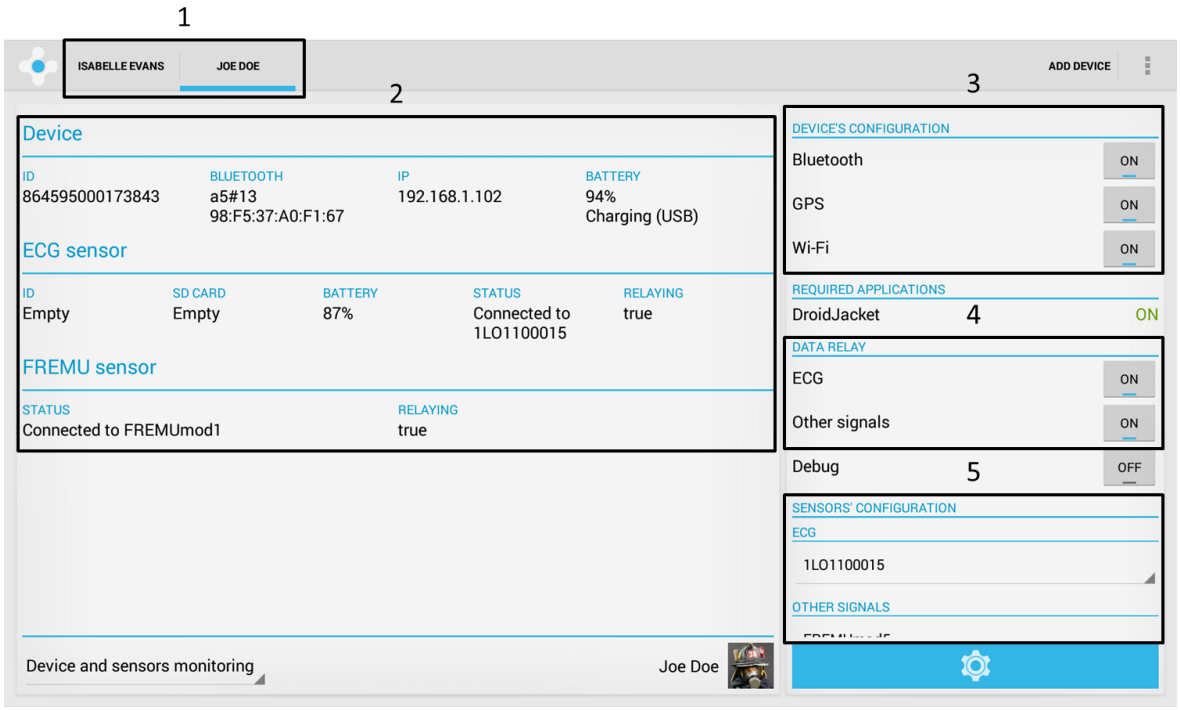


Figure 4.1: Android client application interface developed to configure DroidJacket. This application is able to manage multiple devices (1) and for each device it allows to turn ON or OFF the device own resources (3), enable or disable sensors' data streams (4) and change the Bluetooth MAC address of a sensor type (5). Also, the application provides a diagnostic interface (2) in order to observe the global status of the device and the associated sensors, such as IP and Bluetooth MAC addresses, batteries and if the sensors are connected and relaying.

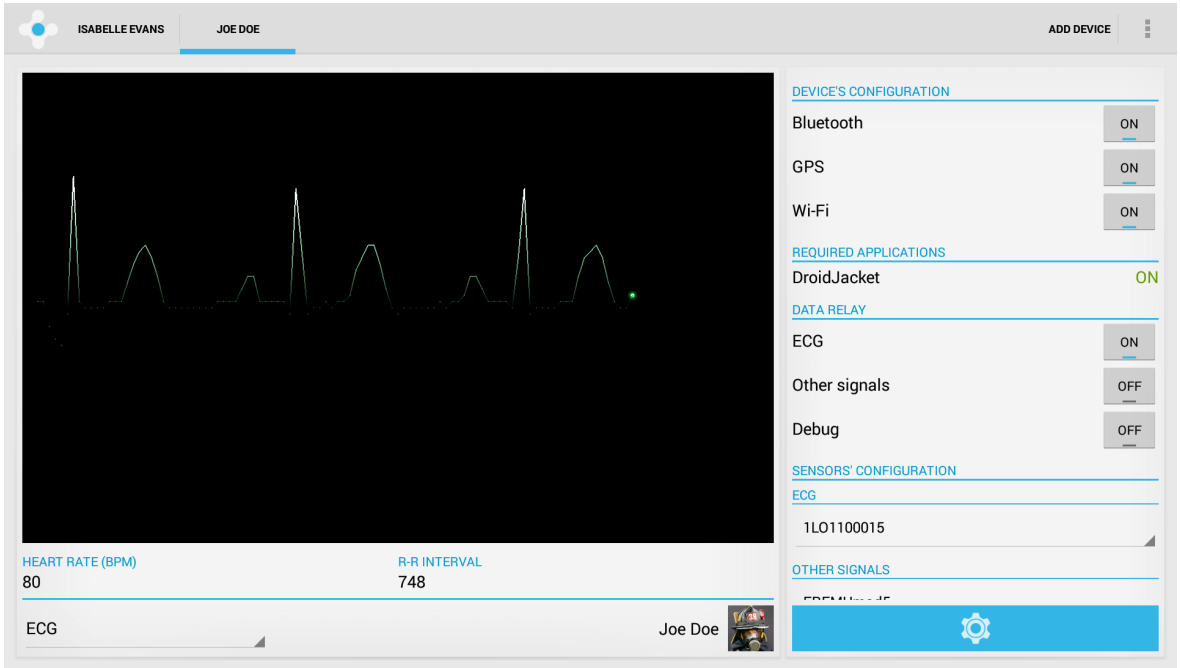
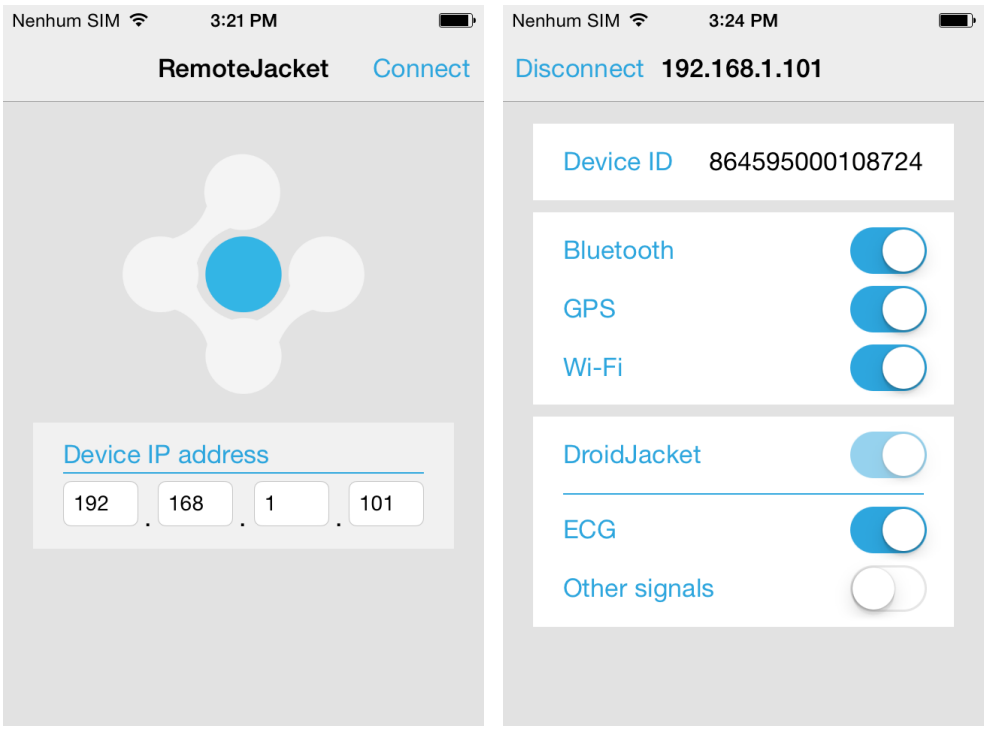


Figure 4.2: Android client application interface developed to monitor the ECG signal, heart rate and R-R interval values provided by DroidJacket.



(a) Initial view.

(b) Configuration view.

Figure 4.3: iOS client application interfaces. This application uses directly the Wi-Fi to connect to device – without Bluetooth handshake. After the user introduces the device's IP address and connects to it the application is able to turn ON or OFF the device own resources and enable or disable the sensors' data streams.

4.4 VITALREMOTE'S EVALUATION

The system was developed over already known and proved technologies such as Bluetooth or Wi-Fi so our focus was on assessing VitalRemote in multiple types of configuration of possible DroidJacket deployments in two possible scenarios extracted from the original VitalResponder scenarios: 1) communication over a standard Wi-Fi network and 2) over an ad-hoc network where each phone was a node of the network (i.e. VitalRemote client device and the devices with DroidJacket and VitalRemote installed). VitalJacket boxes [35] were used as external Bluetooth resources and an ECG provider. During the tests the ECG was acquired at 500Hz from volunteers or using the ECG simulator. Also, the VitalJacket environment sensors (illustrated in Fig. 4.4) were used as environment values provider, such as ambient temperature, atmospheric pressure, Carbon Monoxide concentration and altitude.

In each of the several combinations tested it was used the Handshake and REST interfaces to perform the following operations: turning on/off the Wi-Fi resources, starting DroidJacket and swapping the sensors remotely (VitalJacket boxes and environment units) besides inspecting the incoming ECG signal from the VitalJacket boxes and the environment measurements from the VitalJacket environment sensors. All Wi-Fi communications between devices assume they all are on the same wireless network i.e. VitalRemote client device and devices running VitalRemote service and DroidJacket.

The Table 3 and Table 4 show the diversity of used devices in each test scenario. The devices were chosen in order to have the most different set of devices as possible.

In the tests, besides the android mobile-based deployment (Fig. 4.5.a), a specific attention was given to the Tronsmart T428 – Android PC based – an option to the standard Smartphone (Fig. 4.5.b). Tronsmart T428 is an Android PC that provides both Wi-Fi and Bluetooth interfaces but not a screen, implying the use of its High-Definition Multimedia Interface (HDMI)

Figure 4.4: Environment module that measures ambient temperature, CO concentration, atmospheric pressure and altitude. This module provides a Bluetooth interface in order to access these values.

VitalRemote client	Version
Archos 80 G9 turbo	Android 4.0.4
Google Nexus S	Android 4.4.2
iPhone 4	OS 7.1
Samsung Galaxy Tab 10.1 GT-P7510	Android 3.2
Sony Xperia SGP321	Android 4.3

Table 4.1: List of devices used as VitalRemote client and its OS versions.

VitalRemote service + DroidJacket	Version
Google Nexus S	Android 4.4.2
Optimus Boston Z71	Android 2.3.7
Tronsmart T428	Android 4.2
ZTE TMN A5	Android 2.2

Table 4.2: List of devices, which ran VitalRemote service and DroidJacket, and its OS versions.

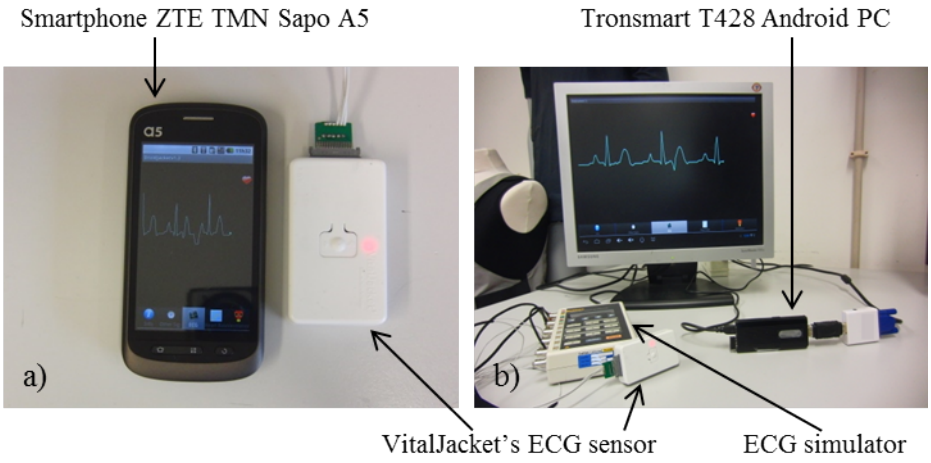


Figure 4.5: Example of setups used in tests: a) Smartphone ZTE TMN Sapo A5 and b) the Android PC Tronsmart T428.

to connect to a LCD monitor to assess the device configuration changes.

The DroidJacket mobile-based scenario was also evaluated under an ad-hoc firefighter MESH network scenario. VitalRemote client performed has expected when configuring DroidJacket nodes deployed on the MESH. In this test, VitalRemote client was also installed in a MESH node.

Using VitalRemote, it was possible to configure DroidJacket. In the case of the Android PC, a LCD monitor was used to confirm the expected behavior of the system namely by observing the incoming ECG data and measures of an environmental sensor.

4.5 VITALREMOTE AND OTHER SYSTEMS

In section 2 we presented some CPS examples which were developed also to configure their own systems. Although VitalRemote was designed to interact with a generic CPS they

have similar characteristics for configuration and management of CPS. Similarly to Smart Gateway and Smart Electricity Meters project, VitalRemote uses a REST interface to access and identify resources as well as to configure them. In this REST interface it is possible to map application specific commands to REST messages allowing, for example, to specify resources parameters, like the number of PPEs and their respective IDs in the PPE Monitoring System, or the type and number of sensor nodes of the WBAN for Health Monitoring project. The proposed VitalRemote interfaces also allows, similarly to Health Monitoring system, the initialization of resources, such as IP addresses and ports, and their customization, e.g. network transport protocol. However, there are other characteristics that are worth discussing.

The Weiss, Mattern, Graml, *et al.* [23] present a CPS project based on smart electricity meters that aims to manage home energy. This project has a mobile application to control the system and uses REST interface to access the Web Server which responds with JSON messages. Comparing with our system, the VitalRemote uses simpler REST messages needing a simple parser and not a generic Web Server. This allows that VitalRemote can be installed in a wider range of devices with a less rigid topology.

Similarly to Akanmu, Anumba, and Messner [17] we developed a system composed by multiple layers, wherein each one has its specific functions: (1) sensing, composed by the abstraction of logic paired sensors and device applications, (2) device, defined by the concept of VitalRemote device, (3) communication, specified by the Handshake protocol and REST interface, (4) contents and application, and (5) actuation, presented in VitalRemote client. In terms of software architecture, VitalRemote also has a similarity with Smart Gateway project [32] (Fig. 2.8 and Fig. 3.2). Both architectures are divided in functional components and VitalRemote device works as a single point to communicate with sensors and actuators as well as gatherer of measured data by them. The same authors demonstrate that JSON as messages codification is a good choice for small and large messages, once XML codification has a worse performance for large messages. VitalRemote uses JSON messages as response of client requests once that messages size can vary from device to device, e.g. depending of their own hardware resources number the size of response to `GET/DEVICE/HARDWARE_RESOURCES` request can be different.

Compared to Personal Protective Equipment monitoring system [34] this project stores the configurations of the all devices in the network into the single Coordinator database, in order to have an easier system administration. In VitalRemote each device contains its own configurations and it is able to return them when requested. In this project as well as

Health Monitoring System, proposed in [5], each person carries a device which is connected to a hardware module responsible to ensure the communication with the rest of system. The last enumerated project uses, besides Internet, mobile telephone networks, such as GPRS, to connect to medical server. In VitalRemote it is a requirement that a device contains, at least, Bluetooth and Wi-Fi technologies to guarantee the communications.

CHAPTER 5

Conclusion and future work

In this work was presented a system, VitalRemote, oriented to support wireless management and configuration of CPS. VitalRemote defines a generic concept of device with resources abstraction that allows any VitalRemote compliant device to be managed and configured wirelessly by the system.

Through a Bluetooth or NFC Handshake interface and a Wi-Fi REST interface any VitalRemote client is able to configure a given VitalRemote device resource. The Handshake was designed for basic configurations in proximity scenarios. The REST interface allows advanced configurations but implies a Wi-Fi connection. It offers the possibility to map custom REST messages to application specific commands to be translated and executed on the VitalRemote device. These commands are used in the configuration of device own hardware resources, pairing of sensor and logical resources (e.g. application providing a VitalRemote configuration interface – in Android based on Content Provider).

VitalRemote current version was implemented in Android OS and tested using several heterogeneous configurations where no direct access to the devices was possible, using the DroidJacket scenario with common constrains found in first responders scenarios namely in wearable sensors under heavy duty and protective garments. By reengineering the original DroidJacket to be a VitalRemote compliant setup, it was possible to manage several different setups namely to configure CPS specific parameters (e.g. data collector IP address), turn on and off resources (e.g. Bluetooth, GPS), exchange transparently hardware sensors while

preserving logical data streams resources (e.g. swap VitalJacket box or environment monitoring modules) or change transmission mode (e.g. TCP, UDP).

The VitalRemote multi-layer architecture abstraction and high level interaction interfaces (Handshake and REST) suggest that our system can be extended to be used in setups other than the DroidJacket and can be adapted to other CPS with different objectives. The VitalRemote point-to-point interface allows it to run without any external resource namely web based servers and does not impose any restriction on the CPS topology.

As future work, VitalRemote could support the ISO/IEEE 11073 standard in order to provide a generic interface with ISO/IEEE 11073 compliant devices that, as in DroidJacket scenario, can provide human monitoring opportunities. This could be achieved by providing an interface for external devices – as already happens with Bluetooth sensors – with the implementation of basic ISO/IEEE 11073 compliant services.

References

[1] M. Weiser, "The Computer for the 21st Century", *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, Jul. 1999, ISSN: 1559-1662. DOI: 10.1145/329124.329126.

[2] M. Conti, S. K. Das, C. Bisdikian, M. Kumar, L. M. Ni, A. Passarella, G. Roussos, G. Tr, ster, G. Tsudik, and F. Zambonelli, "Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber-physical convergence", *Pervasive Mob. Comput.*, vol. 8, no. 1, pp. 2–21, 2012, ISSN: 1574-1192. DOI: 10.1016/j.pmcj.2011.10.001.

[3] I. Lee and O. Sokolsky, "Medical Cyber Physical Systems", in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10, Anaheim, California: ACM, 2010, pp. 743–748, ISBN: 978-1-4503-0002-5. DOI: 10.1145/1837274.1837463. [Online]. Available: http://doi.acm.org/10.1145/1837274.1837463.

[4] M. V. P. Rao and A. Chaganti, "Modeling and Verification of Cyber Physical Systems: Two Case Studies", Indian Institute of Technology Hyderabad.

[5] C. Otto, A. Milenkovi, C. Sanders, and E. Jovanov, "System architecture of a wireless body area sensor network for ubiquitous health monitoring", *J. Mob. Multimed.*, vol. 1, no. 4, pp. 307–326, 2005, ISSN: 1550-4646.

[6] M. F. M. Colunas, J. M. A. Fernandes, I. C. Oliveira, and J. P. S. Cunha, "DroidJacket: An Android-based application for first responders monitoring", in *Information Systems and Technologies (CISTI), 2011 6th Iberian Conference on*, pp. 1–4.

[7] K.-J. Park, R. Zheng, and X. Liu, "Cyber-physical systems: Milestones and research challenges.", *Computer Communications*, vol. 36, no. 1, pp. 1–7, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/comcom/comcom36.html#ParkZL12.

[8] R. Rajkumar, L. Insup, S. Lui, and J. Stankovic, "Cyber-physical systems: The next computing revolution", in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pp. 731–736, ISBN: 0738-100X.

[9] S. Jianhua, W. Jiafu, Y. Hehua, and S. Hui, "A survey of Cyber-Physical Systems", in *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*, pp. 1–6. DOI: 10.1109/WCSP.2011.6096958.

[10] Y. Tan, S. Goddard, and L. C. Perez, "A prototype architecture for cyber-physical systems", *SIGBED Rev.*, vol. 5, no. 1, pp. 1–2, 2008, ISSN: 1551-3688. DOI: 10.1145/1366283.1366309.

[11] A. L. Edward and A. S. Sanjit, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*, 1st ed. Lee and Seshia, 2010, ISBN: 978-0-557-70857-4.

[12] (2012). Cyber-Physical Systems, [Online]. Available: http://cyberphysicalsystems.org/ (visited on 05/14/2014).

[13] F.-J. Wu, Y.-F. Kao, and Y.-C. Tseng, "Review: From wireless sensor networks towards cyber physical systems", *Pervasive Mob. Comput.*, vol. 7, no. 4, pp. 397–413, 2011, ISSN: 1574-1192. DOI: 10.1016/j.pmcj.2011.03.003.

[14] L. Hyun Jung and K. Soo Dong, "A Service-Based Approach to Designing Cyber Physical Systems", in *Computer and Information Science (ICIS), 2010 IEEE/ACIS 9th International Conference on*, pp. 895–900. DOI: 10.1109/ICIS.2010.73.

[15] H. Liang, X. Nannan, K. Zhejun, and Z. Kuo, "Review of Cyber-Physical System Architecture", in *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2012 15th IEEE International Symposium on*, pp. 25–30. DOI: 10.1109/ISORCW.2012.15.

[16] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and W. Shige, "Toward a Science of Cyber-Physical System Integration", *Proceedings of the IEEE*, vol. 100, no. 1, pp. 29–44, 2012, ISSN: 0018-9219. DOI: 10.1109/JPROC.2011.2161529.

[17] A. Akanmu, C. Anumba, and J. Messner, "Scenarios for cyber-physical systems integration in construction", *Journal of Information Technology in Construction (ITcon)*, vol. 18, pp. 240–260, 2013.

[18] (2002-2014). Action webs, [Online]. Available: http://chess.eecs.berkeley.edu/actionwebs/ (visited on 05/23/2014).

[19] R. Baheti and H. Gill, "The Impact of Control Technology", IEEE Control Systems Society, Report, 2011.

[20] M. R. Hafner, D. Cunningham, L. Caminiti, and D. Del Vecchio, "Automated Vehicle-to-Vehicle Collision Avoidance at Intersections", 18th ITS World CongressTransCoreITS AmericaERTICO - ITS EuropeITS Asia-PacificOrlando,Florida,USA StartDate:20111016 EndDate:20111020 Sponsors:TransCore, ITS America, ERTICO - ITS Europe, ITS Asia-Pacific, p. 11.

[21] R. Mangharam, D. Weller, R. Rajkumar, P. Mudalige, and B. Fan, "GrooveNet: A Hybrid Simulator for Vehicle-to-Vehicle Networks", in *Mobile and Ubiquitous Systems - Workshops, 2006. 3rd Annual International Conference on*, pp. 1–8. DOI: 10.1109/MOBIQW.2006.361773.

[22] H. Kim, S. K. Lee, H. Kim, and H. Kim, "Implementing home energy management system with UPnP and mobile applications", *Computer Communications*, vol. 36, no. 1, pp. 51–62, 2012, ISSN: 0140-3664. DOI: 10.1016/j.comcom.2012.01.007.

[23] M. Weiss, F. Mattern, T. Graml, T. Staake, and E. Fleisch, *Handy feedback: connecting smart meters with mobile phones*, Conference Paper, 2009. DOI: 10.1145/1658550.1658565.

[24] (2014). Json, [Online]. Available: http://www.json.org/ (visited on 05/20/2014).

[25] M. F. M. Colunas, J. M. A. Fernandes, I. C. Oliveira, and J. P. S. Cunha, "Droid Jacket: Using an Android based smartphone for team monitoring", in *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pp. 2157–2161. DOI: 10.1109/IWCMC.2011.5982868.

[26] "High-Confidence Medical Devices: Cyber-Physical Systems for 21st Century Health Care", High Confidence Software and Systems Coordinating Group, Report, 2009.

[27] J. Goldman, R. Schrenker, J. Jackson, and S. Whitehead, "Plug-and-Play in the Operating Room of the Future", *Biomedical Instrumentation & Technology*, pp. 194–199, 2005.

[28] S. Lui, S. Gopalakrishnan, L. Xue, and W. Qixin, "Cyber-Physical Systems: A New Frontier", in *Sensor Networks, Ubiquitous and Trustworthy Computing, 2008. SUTC '08. IEEE International Conference on*, pp. 1–9. DOI: 10.1109/SUTC.2008.85.

[29] L. Schmitt, T. Falck, F. Wartena, and D. Simons, "Novel ISO/IEEE 11073 Standards for Personal Telehealth Systems Interoperability", in *High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, 2007. HCMDSS-MDPnP. Joint Workshop on*, pp. 146–148. DOI: 10.1109/HCMDSS-MDPnP.2007.9.

[30] J. Yao and S. Warren, "Applying the ISO/IEEE 11073 Standards to Wearable Home Health Monitoring Systems", *Journal of Clinical Monitoring and Computing*, vol. 19, no. 6, pp. 427–436, 2005, ISSN: 1387-1307. DOI: 10.1007/s10877-005-2033-7.

[31] M. Clarke, D. Bogia, K. Hassing, L. Steubesand, T. Chan, and D. Ayyagari, "Developing a Standard for Personal Health Devices based on 11073", in *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pp. 6174–6176, ISBN: 1557-170X. DOI: 10.1109/IEMBS.2007.4353764.

[32] Q. Li, W. Qin, B. Han, R. Wang, and L. Sun, "A case study on REST-style architecture for cyber-physical systems: Restful smart gateway", *Comput. Sci. Inf. Syst.*, pp. 1317–1329, 2011.

[33] J. Zheng and M. J. Lee, "A comprehensive performance study of IEEE 802.15.4", in. IEEE Press, Wiley Interscience, 2006, ch. 4.

[34] S. Barro-Torres, T. M. Fernandez-Carames, H. J. Perez-Iglesias, and C. J. Escudero, "Real-time personal protective equipment monitoring system", *Computer Communications*, vol. 36, no. 1, pp. 42–50, 2012, ISSN: 0140-3664. DOI: 10.1016/j.comcom.2012.01.005.

[35] J. P. S. Cunha, B. Cunha, A. S. Pereira, W. Xavier, N. Ferreira, and L. Meireles, "Vital-Jacket: A wearable wireless vital signs monitor for patients' mobility in cardiology and sports", in *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2010 4th International Conference on*, pp. 1–2. DOI: 10.4108/ICST.PERVASIVEHEALTH2010.8991.

[36] P. R. L. Azevedo, "VitalAir - suporte a monitorização online da respiração para bombeiros", Master's Thesis, 2012.

[37] F. M. O. Marques, "Fireman: sistema android para gestão respiratória de bombeiros em equipa", Master's Thesis, 2013.

[38] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., 2003, p. 256, ISBN: 0321193687.

[39] (2014). Android Services, [Online]. Available: http://developer.android.com/guide/components/services.html (visited on 02/24/2014).